Graph Convolutional Networks

Sergey Ivanov

Ph.D.

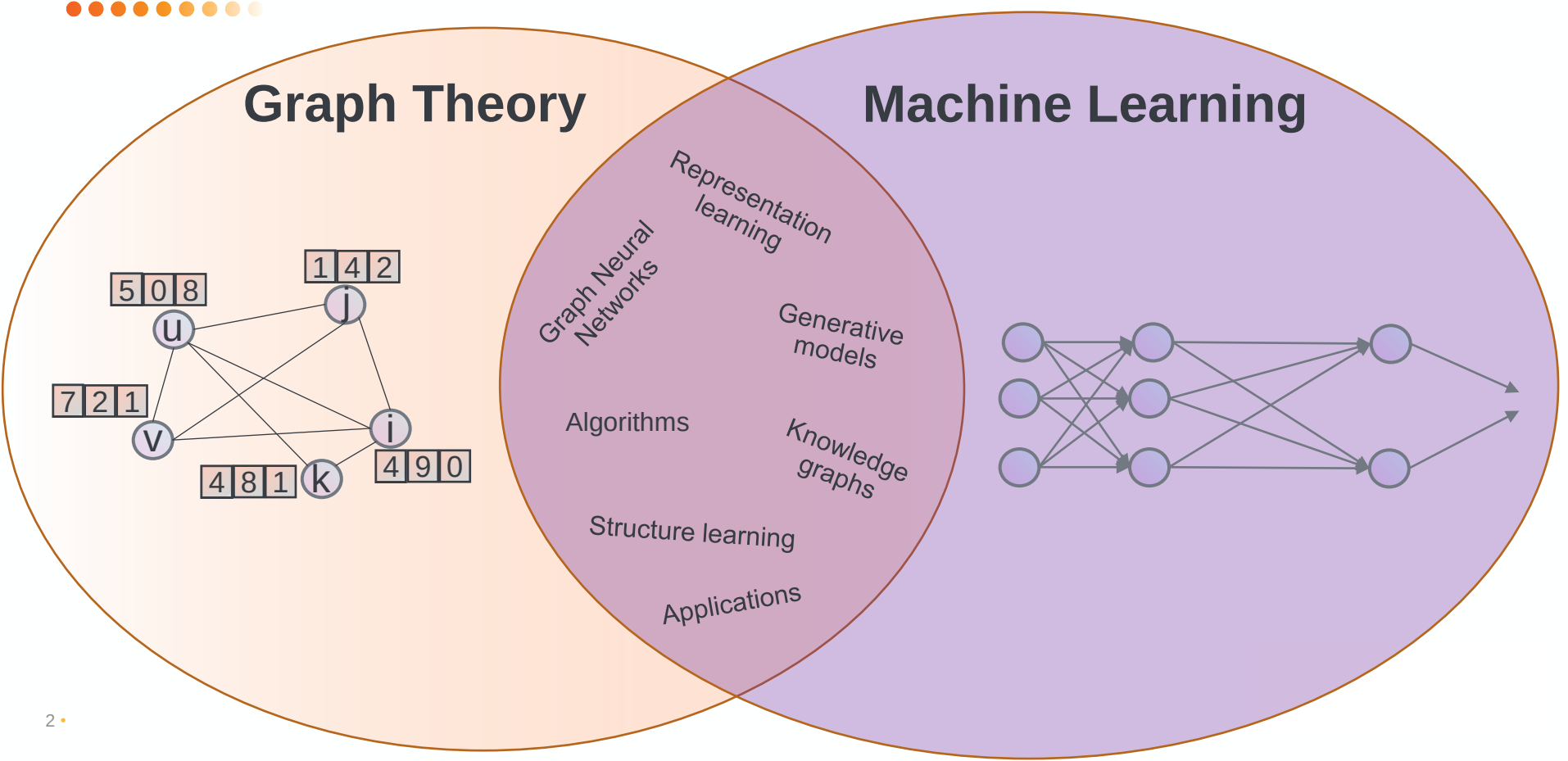
Research Scientist, Criteo

@graphML

@SergeyI49013776

Slides are provided by Xavier Bresson for NYU Deep Learning Course 2020

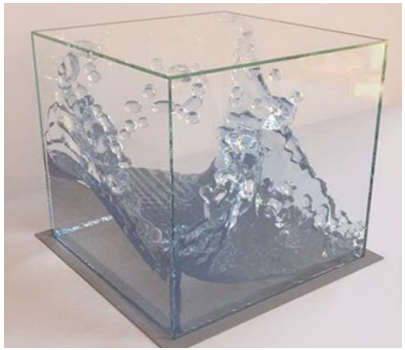
What is it all about?



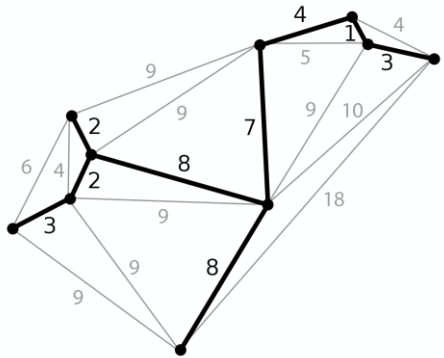
What are successful stories?

Drug discovery

Particle dynamics



Combinatorial optimization



Self-driving cars

[1] A Deep Learning Approach to Antibiotic Discovery, Stokes et al. 2020
[2] Learning to Simulate Complex Physics with Graph Networks, Sanchez-Gonzalez et al. 2020
[3] Reinforcement Learning for Combinatorial Optimization: A Survey, Mazyavkina et al. 2020
[4] Spatially-Aware Graph Neural Networks for Relational Behavior Forecasting from Sensor Data, Casas et al. 2020

Community

Conferences

NEURAL INFORMATION PROCESSING SYSTEMS

ICML International Conference On Machine Learning

KDD2020

ICLR

Workshops

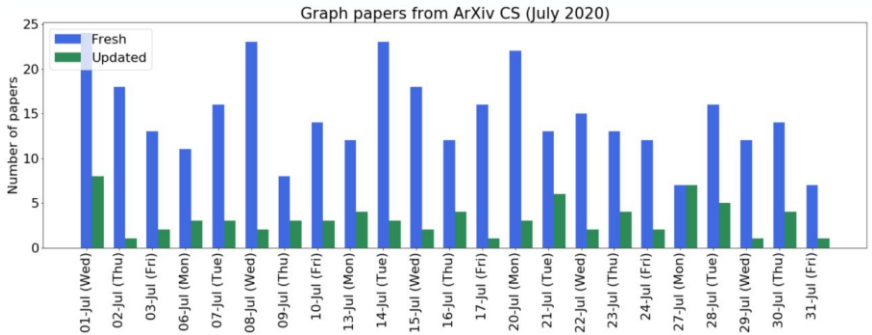
BRIDGE BETWEEN PERCEPTION AND REASONING: GRAPH NEURAL NETWORKS & BEYOND

GRL+

DL4G@WWW2020

Mining and Learning with Graphs

ArXiv



Social

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: Benchmarking Graph Neural Networks
- Conclusion

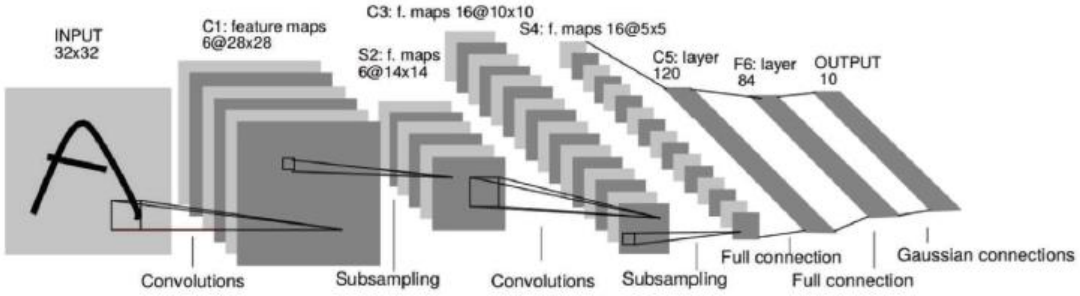
ConvNets

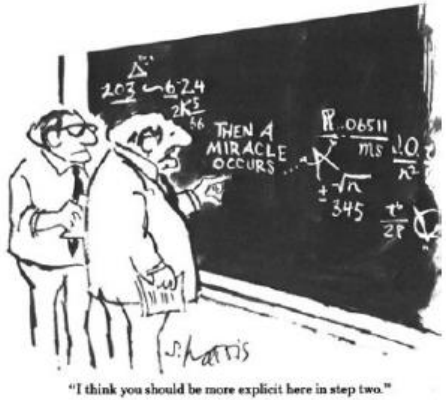
- ConvNets are powerful architectures to solve high-dimensional learning problems.

- Curse of dimensionality :

  $\dim(\text{image}) = 1024 \times 1024 \approx 10^6$

  For N=10 samples/dim $\Rightarrow 10^{1,000,000}$ points

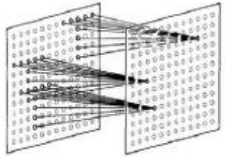


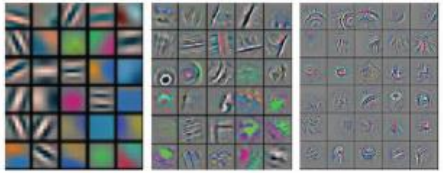


"I think you should be more explicit here in step two."

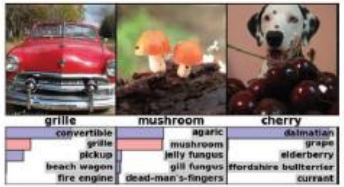
ConvNets

- 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)

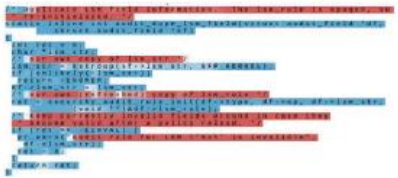




- ConvNets leverage the compositionality structure :
  - They extract compositional features and feed them to classifier, recommender, etc (end-to-end systems).



Computer Vision



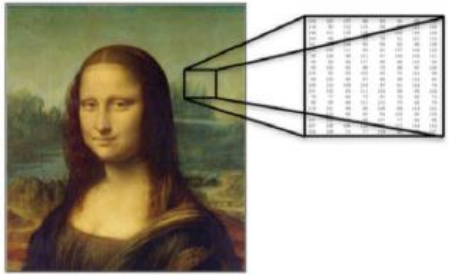
NLP

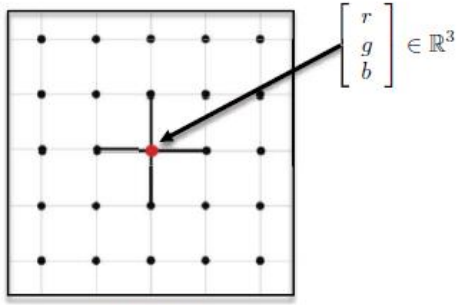
Speech

Game of Go

Data Domain

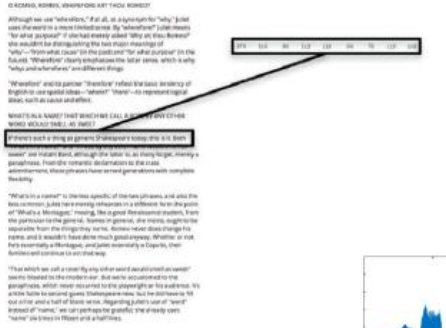
- Images, volumes, videos lie on
  2D, 3D, 2D+1 Euclidean domains (grids)

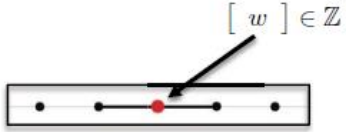




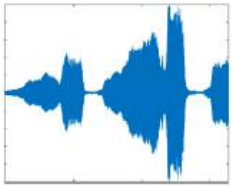
2D grid

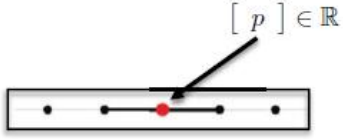
- Sentences, words, speech lie on
  1D Euclidean domain





1D grid





1D grid

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

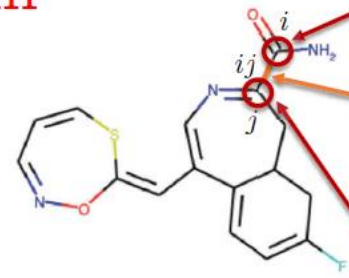
Graph Domain

Social networks
(Advertisement/
recommendation)

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

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

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



Quantum Chemistry
(novel molecules for drugs
and materials)

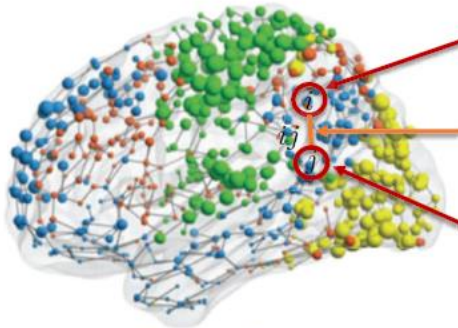
$\textbf{Atom}_i \begin{bmatrix} \text{type} \\ \text{coordinates} \\ \text{charge} \end{bmatrix}_i \in \mathbb{R}^{d_v}$

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

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

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

Brain connectivity
(sMRI)



Brain analysis
(Neuroscience/neuro-diseases)

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

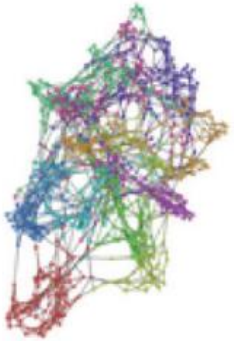
$\textbf{Cerebral connection}_{ij} \quad A_{ij} \in \mathbb{R}_+$

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

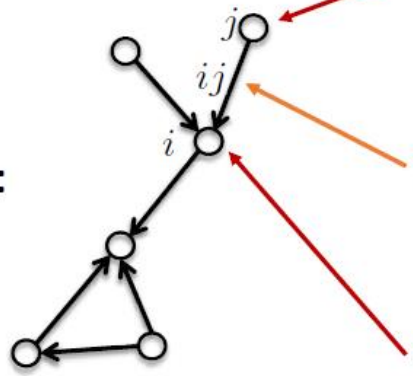
Functional
activations (fMRI)

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)

$$\mathcal{G} = (V, E, A)$$

$$V = \{1, .., n\}$$

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

Convolution

- **Convolutional layer** (for grids) :

$$h^{\ell+1} = w^{\ell} * h^{\ell}$$

$$n_1 \times n_2 \times d \qquad n_1 \times n_2 \times d$$

$$3 \times 3 \times d$$

h^{ℓ}

Image/Hidden features

$*$

w^{ℓ}

Pattern/kernel (learned by backpropagation)

$=$

$h^{\ell+1}$

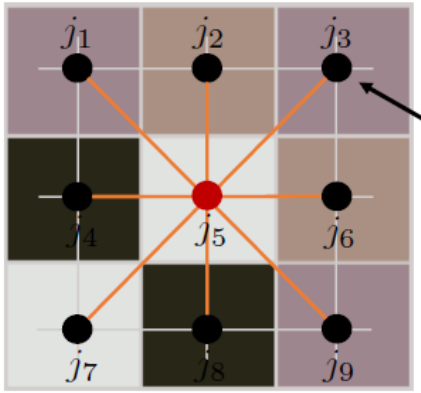
Convolution

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

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

$3 \times 3 \times d$

$n_1 \times n_2 \times d$



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

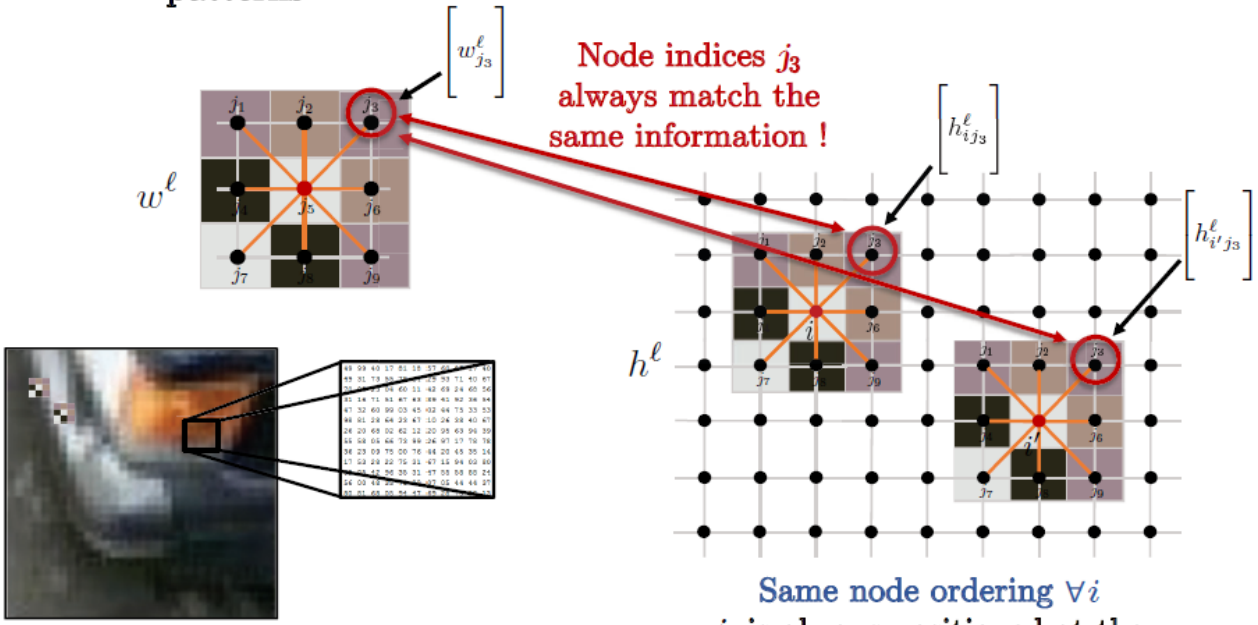
All nodes of the template w^l are always ordered/positioned the same way !

w^ℓ

Convolution

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

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



Same node ordering $\forall i$

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

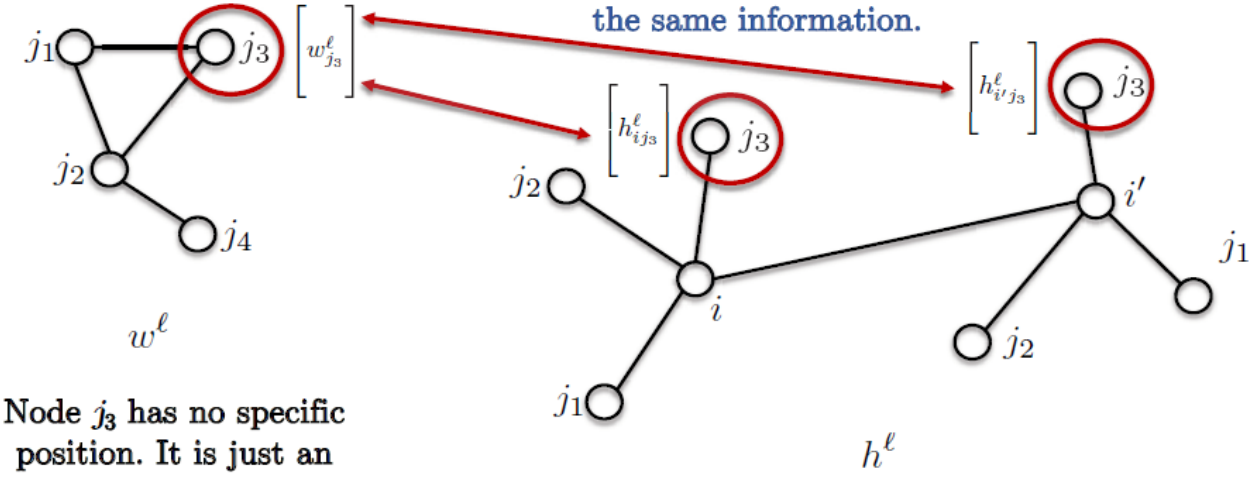
$$\left\langle \begin{bmatrix} w_{j_3}^\ell \end{bmatrix}, \begin{bmatrix} h_{ij_3}^\ell \end{bmatrix} \right\rangle$$

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

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 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.

$$\langle \left[w^\ell_{j_3} \right], \left[h^\ell_{ij_3} \right] \rangle$$

$$\langle \left[w^\ell_{j_3} \right], \left[h^\ell_{i'j_3} \right] \rangle$$

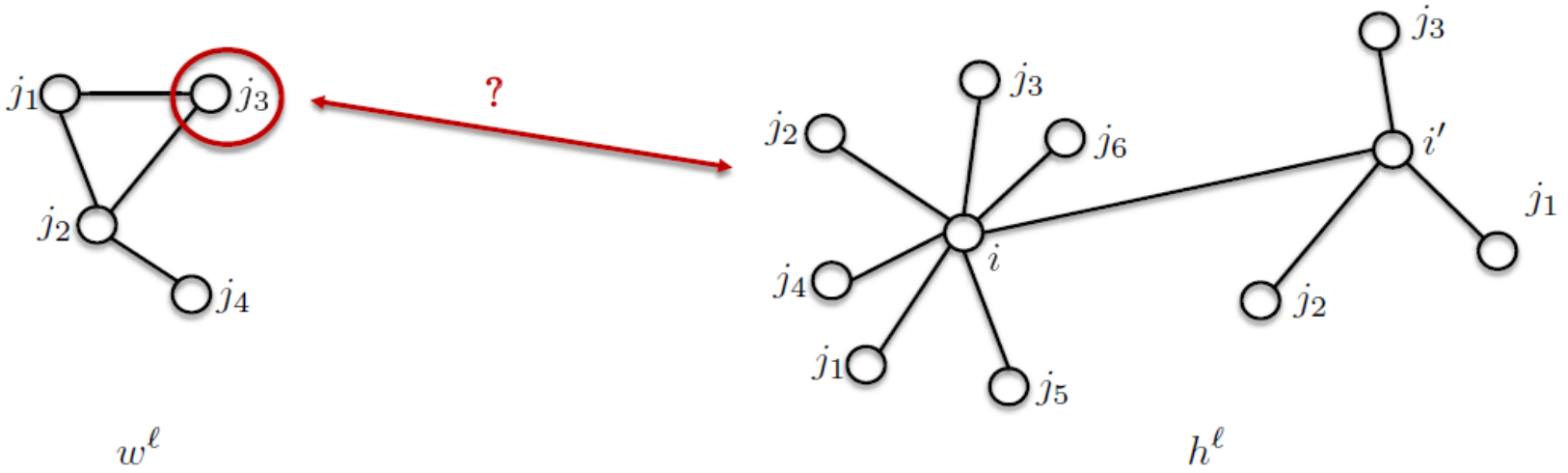
These matching scores have no meaning as they do not compare the same information.

$$h_i^{\ell+1} = w^\ell *_{\mathcal{G}} h_i^\ell = \sum_{j \in \mathcal{N}_i} \langle w^\ell_j, h^\ell_{ij} \rangle$$

$n \times d$

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

Graph Laplacian

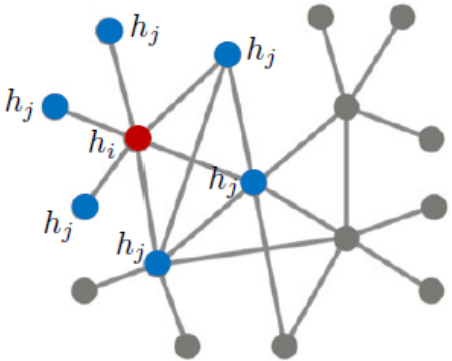
- Core operator in Spectral Graph Theory

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

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

- 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

Lap Eigenvectors/ Fourier functions

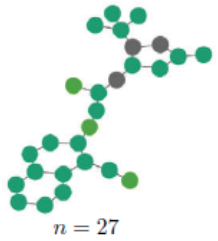
$$\underset{n \times n}{\Delta} = \Phi^T \Lambda \Phi$$

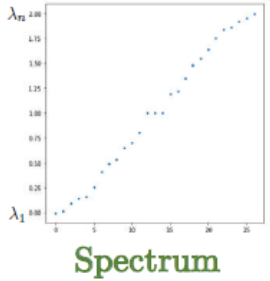
Fourier functions form an orthonormal basis

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

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

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





Spectrum

Fourier Transform

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

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

Fourier transforms are one line of code (linear operations)

$$
\begin{cases}
\underset{n\times 1}{\mathcal{F}(h)} = \Phi^T h = \hat{h} & \text{Fourier Transform/ coefficients of Fourier Series} \\
\underset{n\times 1}{\mathcal{F}^{-1}(\hat{h})} = \Phi \hat{h} & \text{Inverse Fourier Transform} \\
\qquad = \Phi\Phi^T h = h \text{ as } \mathcal{F}^{-1} \circ \mathcal{F} = \Phi\Phi^T = \mathrm{I} & \text{Orthonormal basis/ Invertible transformation}
\end{cases}
$$

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

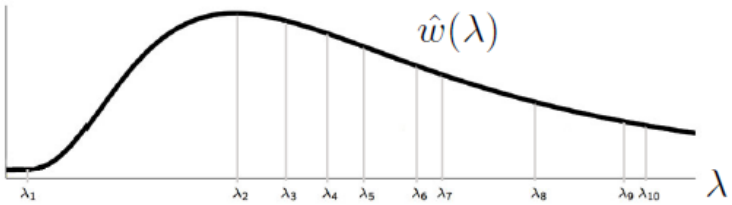
Convolution Theorem

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

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

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

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



Expensive computation $O(n^2)$

No FFT

Spectral function/filter

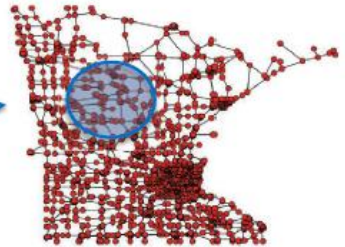
Vanilla Spectral GCN[1]

- Graph spectral convolutional layer :

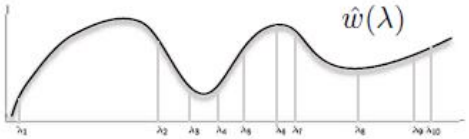
$$
\begin{aligned}
h^{\ell+1}_{n\times d} &= \eta(\, w^{\ell} * h^{\ell} \,) \\
&= \eta(\, \hat{w}^{\ell}(\Delta) h^{\ell} \,) \\
&= \eta(\, \underset{n\times n}{\Phi \hat{w}^{\ell}(\Lambda) \Phi^{T}} \underset{n\times d}{h^{\ell}} \,)
\end{aligned}
$$

$n \times n$

Spatial filter



Spectral filter



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

- 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 :

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

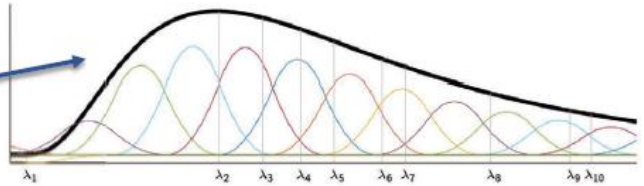
$$\hat{w}^\ell(\Lambda)_{n\times n} = \mathrm{diag}(B w^\ell)$$

B: $n \times n$ (from $n \times K$ and $K \times 1$)

The K coefficients w^l are learned by backpropagation

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

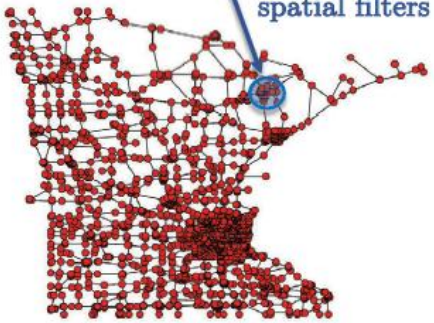
Smooth spectral filters



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

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

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)

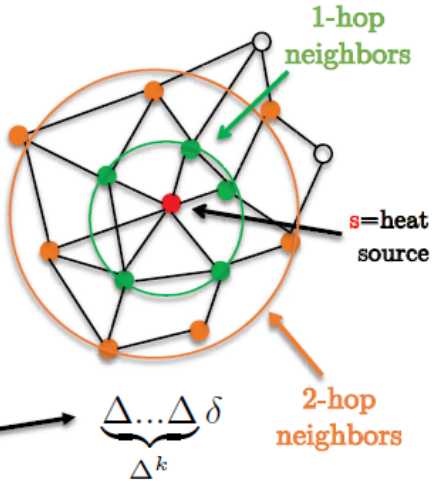
[1] Bruna, Zaremba, Szlam, LeCun, Spectral Networks and Locally Connected Networks on Graphs, 2014
[2] Henaff, Bruna, LeCun, Deep Convolutional Networks on Graph-Structured Data, 2015

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 :

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

Full matrix



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

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

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

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

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 :

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

Recursive equation

- Sequence $\{X_k\}$ is generated by multiplying a matrix Δ and a vector X_{k-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 :

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

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

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

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

- 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)

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

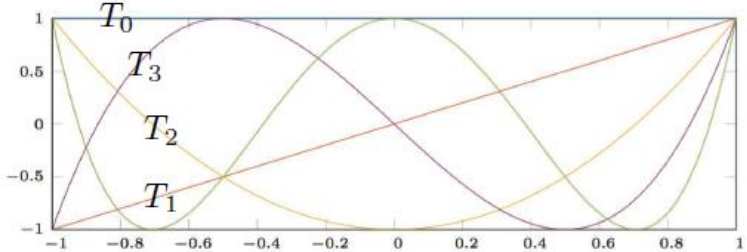
Chebyshev Polynomials

- 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

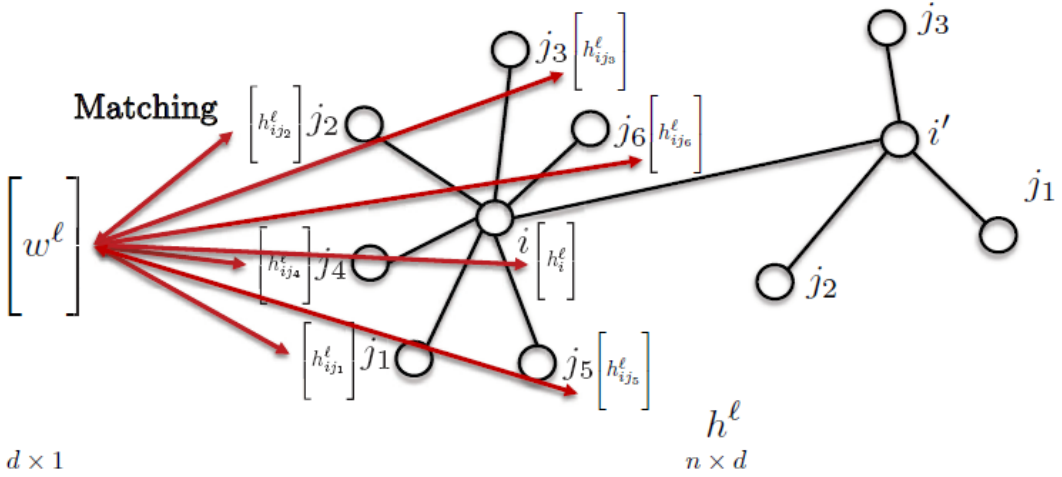
- Filters are exactly localized in K-hop support
- O(*1*) parameters to learn per layer
- O(n) learning complexity
- Stable under coefficients perturbation



Orthonormal basis

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**

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

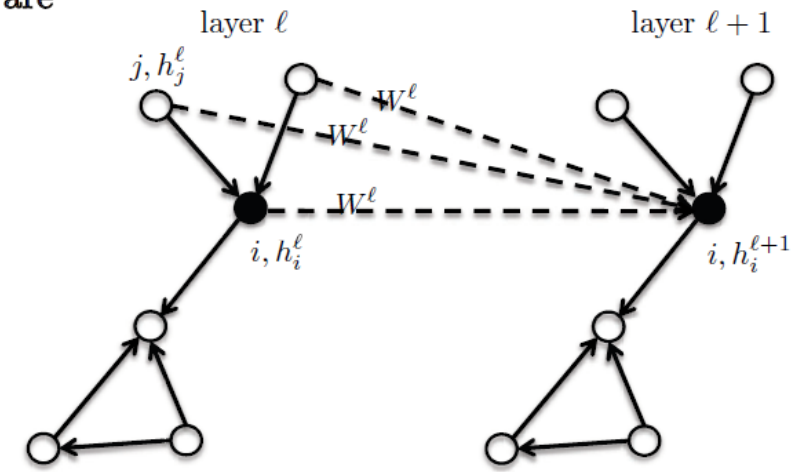
Vectorial representation

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

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

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



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

[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

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

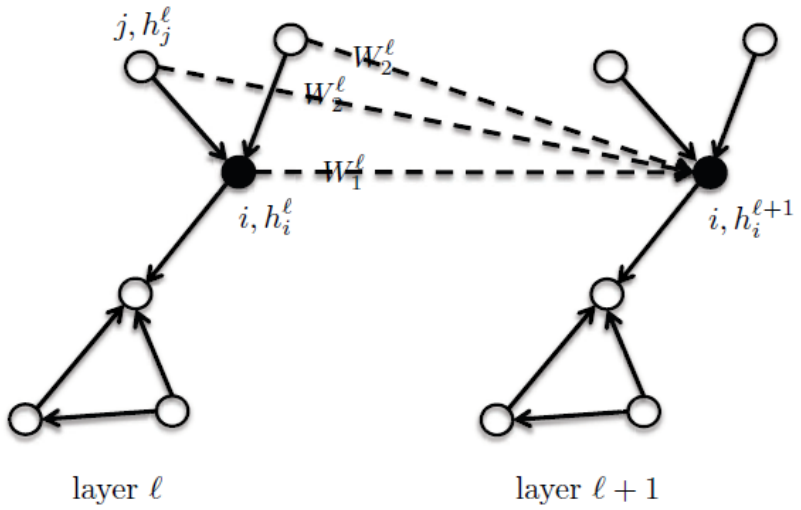
$$\underset{d \times 1}{h_i^{\ell+1}} = \eta\Big(\underset{d \times d}{W_1^\ell} \underset{d \times 1}{h_i^\ell} + \underbrace{\frac{1}{d_i} \sum_{j \in \mathcal{N}_i} \underset{d \times d}{W_2^\ell} \underset{d \times 1}{h_j^\ell}}_{} \Big)$$

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

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

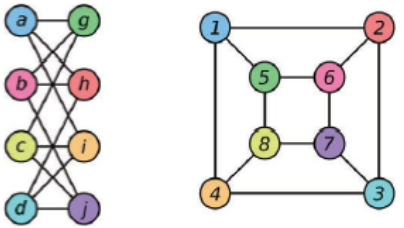
Graph Isomorphism Networks[1] (GIN)

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

- Isotropic GCNs

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

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



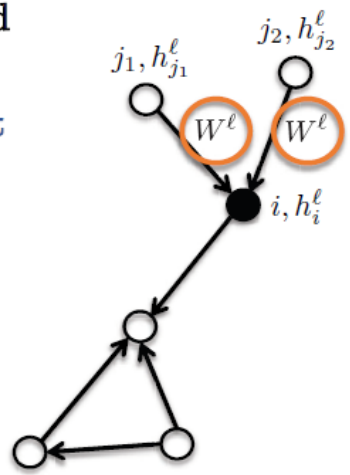
Example of two isomorphic graphs

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

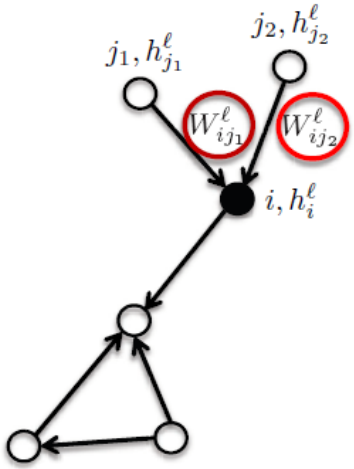
Anisotropic GCNs

- 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

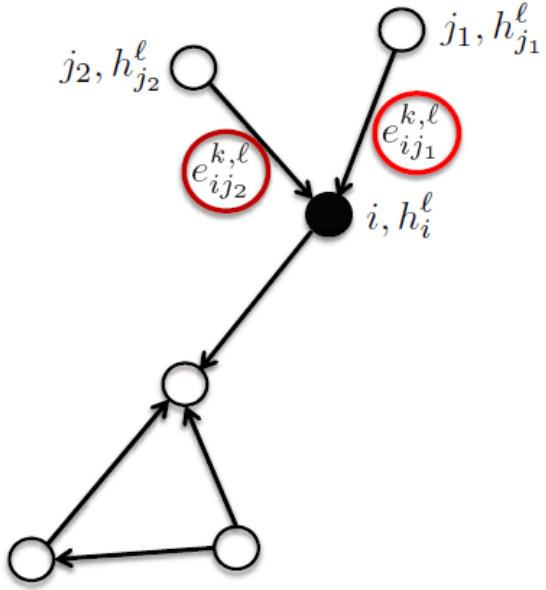
MoNets[1]

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

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

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

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



[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

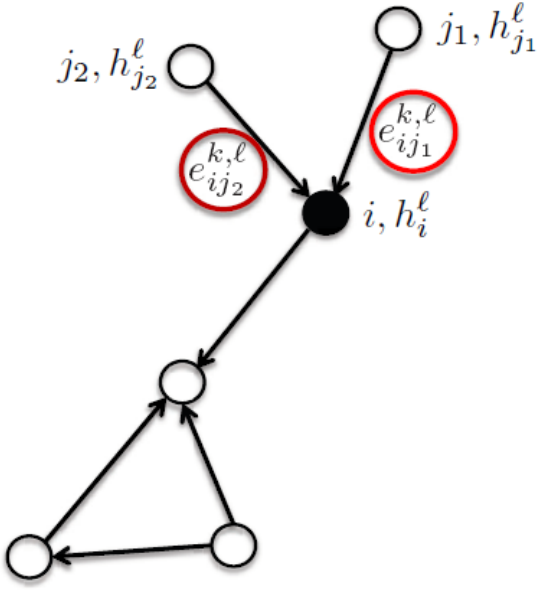
Graph Attention Networks[1] (GAT)

- 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].

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

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

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



[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

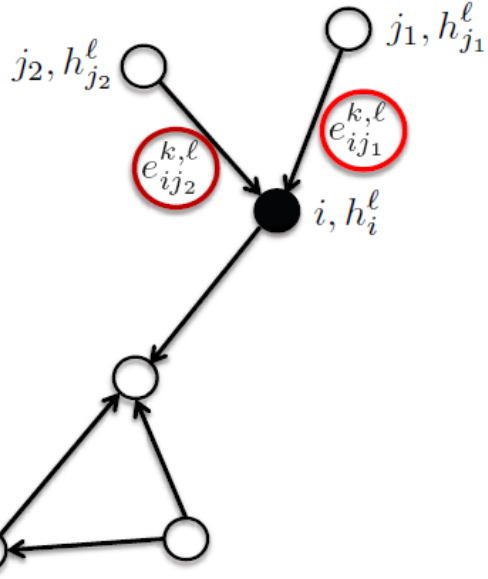
Gated Graph ConvNets[1]

- 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.

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

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

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



[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] :

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

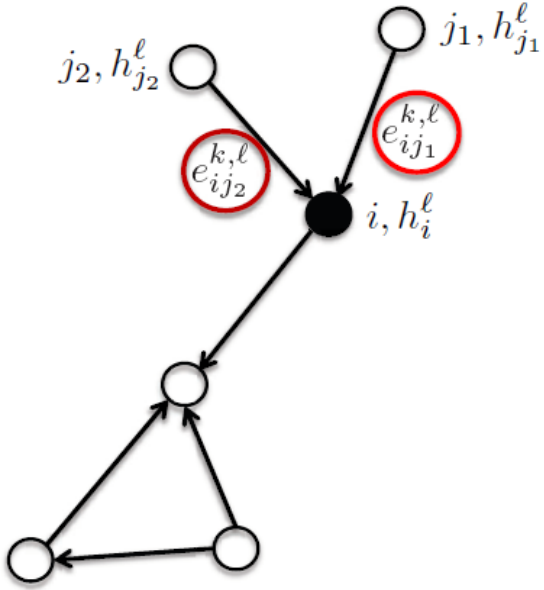
$h_i^{\ell+1}$: $d \times 1$; W^{ℓ}: $d \times d$; $V^{k,\ell} \, h_j^{\ell}$: $\frac{d}{K} \times 1$; $e_{ij}^{k,\ell}$: scalar; Value

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

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

$Q^{\ell,k} h_i^{\ell}$: $1 \times d$; $K^{\ell,k} h_j^{\ell}$: $d \times 1$; Query; Key

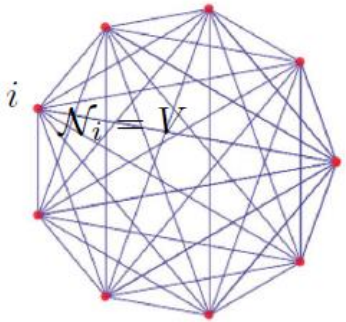
Attention mechanism in 1-hop neighborhood



[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

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

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

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

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

$$h^{\ell+1} = \mathrm{Concat}_{k=1}^{K}\Big(\mathrm{Softmax}(Q^\ell {K^\ell}^T)V^\ell\Big)\ W^\ell$$

$h^{\ell+1}$: $n \times d$; $\mathrm{Softmax}(Q^\ell {K^\ell}^T)V^\ell$: $n \times \frac{d}{K}$; W^ℓ: $d \times d$

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

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

$$V^\ell = h^\ell\ W_V^\ell$$

Q^ℓ: $n \times \frac{d}{K}$; h^ℓ: $n \times d$; W_Q^ℓ: $d \times \frac{d}{K}$

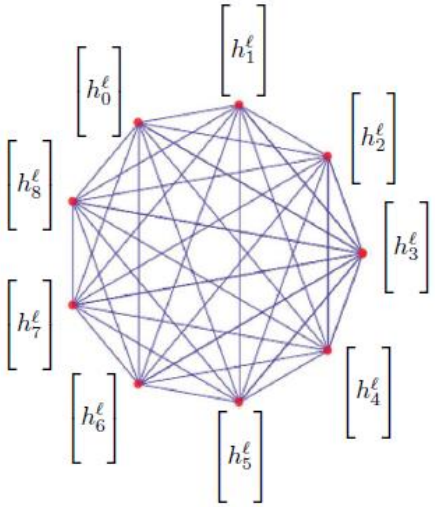
$\mathrm{Softmax}(Q^\ell {K^\ell}^T)$: $n \times \frac{d}{K}$ $\frac{d}{K} \times n$ $= n \times n$

V^ℓ: $n \times \frac{d}{K}$

[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

$\Leftrightarrow$

h_0^ℓ h_1^ℓ h_2^ℓ h_3^ℓ h_4^ℓ h_5^ℓ h_6^ℓ h_7^ℓ h_8^ℓ

Sets of features

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.

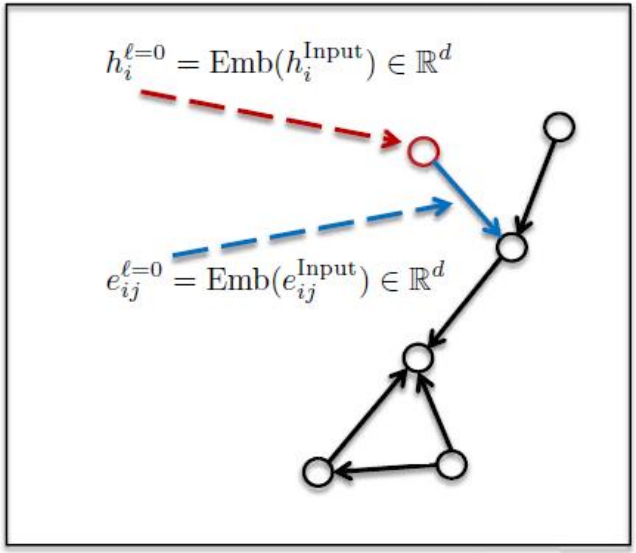
GNN Pipeline

- 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

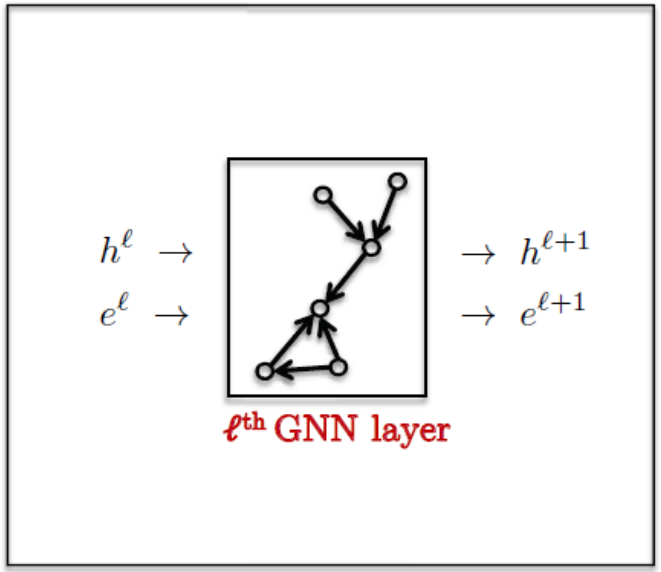
Input features embedded into d-dim spaces, and passes to the GNN layers.

GNN Pipeline

- GNN layers :

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

$\Rightarrow$



$\Rightarrow$

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

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

GNN layers applied L times

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

GNN Pipeline

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

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

  - **Node-level prediction :**

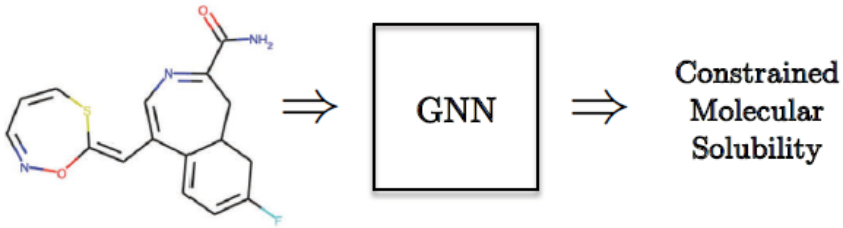
$$h_i^{\ell=L} \in \mathbb{R}^d \quad \Rightarrow \quad \boxed{\text{MLP}} \quad \Rightarrow \quad \text{score}_i \in \begin{cases} \mathbb{R} & \text{regression} \\ \mathbb{R}^K,\ K>1 & \text{classification} \end{cases}$$

  - **Edge-level prediction :**

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

Quantum Chemistry

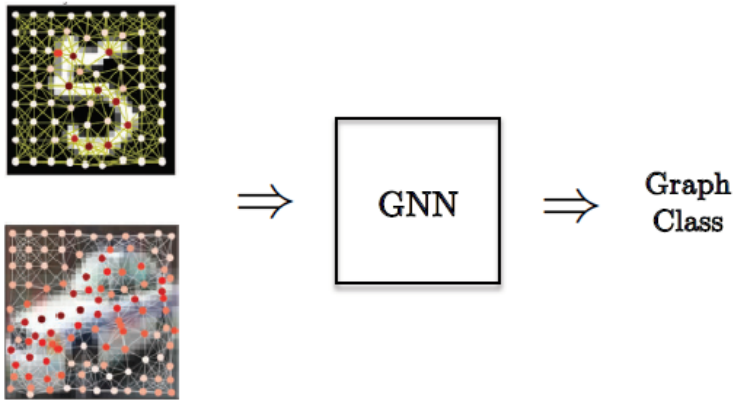
- Graph regression task. We used the ZINC dataset to regress a molecular property known as the constrained solubility [Jin et al., 2018]. Statistics : 10,000 train/1,000 validation/1,000 test graphs of sizes 9-37 nodes/atoms.



Model	#Param	Acc/MAE	Epoch/Total
MLP	108975	0.710±0.001	1.19s/0.02hr
MLP (Gated)	106970	0.681±0.005	1.16s/0.03hr
GCN	103077	0.469±0.002	3.02s/0.08hr
GraphSage	105031	0.429±0.005	3.24s/0.10hr
GIN	103079	0.414±0.009	2.49s/0.06hr
DiffPool	110561	0.466±0.006	12.41s/0.34hr
GAT	102385	0.463±0.002	20.97s/0.56hr
MoNet	106002	0.407±0.007	11.69s/0.28hr
GatedGCN	105735	0.437±0.008	6.36s/0.17hr

Computer Vision

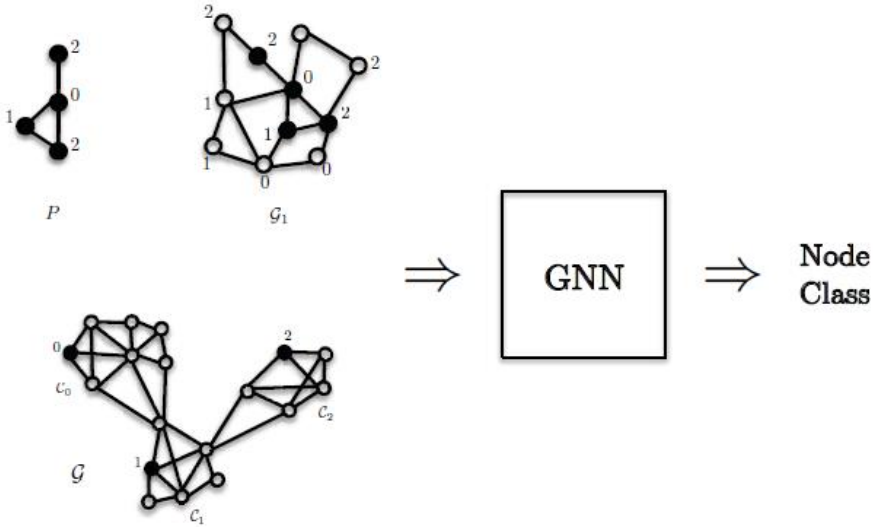
- Graph classification task. We used popular MNIST and CIFAR10 image datasets. Images are converted to graphs using super-pixels [Achanta et al., 2012]. MNIST has 55,000 train/5,000 validation/10,000 test graphs of sizes 40-75 nodes and CIFAR10 has 45,000 train/5,000 validation/10,000 test graphs of sizes 85-150 nodes.



Dataset	Model	#Param	Acc	Epoch/Total
MNIST	MLP	104044	94.46±0.28	21.82s/1.02hr
	MLP (Gated)	105717	95.18±0.18	22.43s/0.73hr
	GCN	101365	89.99±0.15	78.25s/1.81hr
	GraphSage	102691	97.09±0.02	75.57s/1.36hr
	GIN	105434	93.91±0.63	34.30s/0.73hr
	DiffPool	106538	95.02±0.42	170.55s/4.26hr
	GAT	110400	95.62±0.13	375.71s/6.35hr
	MoNet	104049	90.36±0.47	581.86s/15.31hr
	GatedGCN	104217	97.37±0.06	128.39s/2.01hr
CIFAR10	MLP	104044	56.01±0.90	21.82s/1.02hr
	MLP (Gated)	106017	56.78±0.12	27.85s/0.68hr
	GCN	101657	54.46±0.10	100.91s/2.73hr
	GraphSage	102907	65.93±0.30	96.67s/1.88hr
	GIN	105654	53.28±3.70	45.29s/1.24hr
	DiffPool	108042	57.99±0.45	298.06s/10.17hr
	GAT	110704	65.40±0.38	389.40s/7.32hr
	MoNet	104229	53.42±0.43	836.32s/22.45hr
	GatedGCN	104357	69.19±0.28	146.80s/2.48hr

Stochastic Block Models

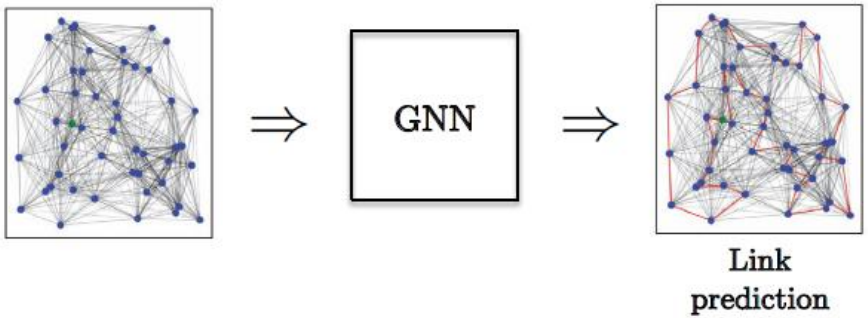
- **Node classification task.** We used the stochastic block model to generate graph patterns embedded in larger graphs (PATTERN dataset) and generate clusters with variable sizes (CLUSTER dataset). Statistics : PATTERN has 10,000 train/2,000 validation/2,000 test graphs of sizes 50-180 nodes. CLUSTER has 10,000 train/1,000 validation/1,000 test graphs of sizes 40-190 nodes.



Dataset	Model	#Param	Acc	Epoch/Total
PATTERN	MLP	105263	50.13±0.00	8.68s/0.10hr
	MLP (Gated)	103629	50.13±0.00	9.78s/0.12hr
	GCN	100923	74.36±1.59	97.37s/2.06hr
	GraphSage	98607	78.20±3.06	79.19s/2.57hr
	GIN	100884	96.98±2.18	14.12s/0.32hr
	GAT	109936	90.72±2.04	229.76s/5.73hr
	MoNet	103775	95.52±3.74	879.87s/21.80hr
	GatedGCN	104003	95.05±2.80	115.55s/2.46hr
CLUSTER	MLP	106015	20.97±0.01	6.54s/0.08hr
	MLP (Gated)	104305	20.97±0.01	7.37s/0.09hr
	GCN	101655	47.82±4.91	66.58s/1.26hr
	GraphSage	99139	44.89±3.70	54.53s/1.05hr
	GIN	103544	49.64±2.09	11.60s/0.27hr
	GAT	110700	49.08±6.47	158.23s/4.08hr
	MoNet	104227	45.95±3.39	635.77s/15.32hr
	GatedGCN	104355	54.20±3.58	81.39s/2.26hr

Combinatorial Optimization

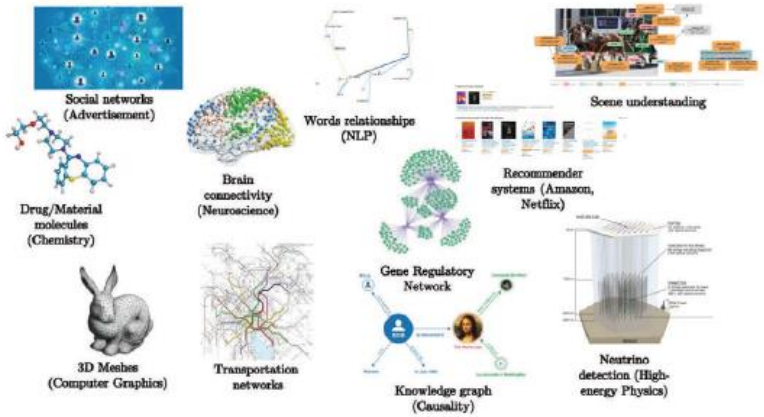
- Edge classification task. We used the Travelling Salesman Problem (TSP). Given a 2D graph, find a tour with minimal length. Statistics : We create train, validation and test sets of 10,000 train TSP / 1,000 TSP / 1,000 TSP, where the number of nodes is randomly selected in [50, 500].



Model	#Param	F1	Epoch/Total
k-NN Heuristic	k=2	F1: 0.693	
MLP	94394	0.548±0.003	53.92s/2.85hr
MLP (Gated)	115274	0.548±0.001	54.39s/2.44hr
GCN	108738	0.627±0.003	163.36s/11.26hr
GraphSage	98450	0.663±0.003	145.75s/16.05hr
GIN	118574	0.655±0.001	73.09s/5.44hr
GAT	109250	0.669±0.001	360.92s/30.38hr
MoNet	94274	0.637±0.010	1433.97s/41.69hr
GatedGCN	94946	0.794±0.004	203.28s/15.47hr

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)

Thank you!