CE7454 : Deep Learning for Data Science

Lecture 14: Graph Neural Networks

Semester 1 2019/20

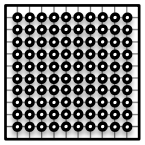
Xavier Bresson

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

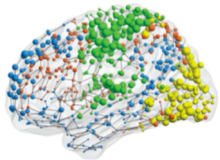
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Outline

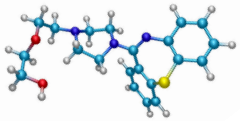
- Part 1: Traditional ConvNets



- Part 2: Spectral Graph ConvNets



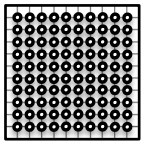
- Part 3: Spatial Graph ConvNets



- Conclusion

Outline

- **Part 1: Traditional ConvNets**



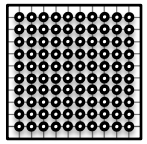
- Part 2: Spectral Graph ConvNets

- Part 3: Spatial Graph ConvNets

- Conclusion

Outline

- Part 1: Traditional ConvNets



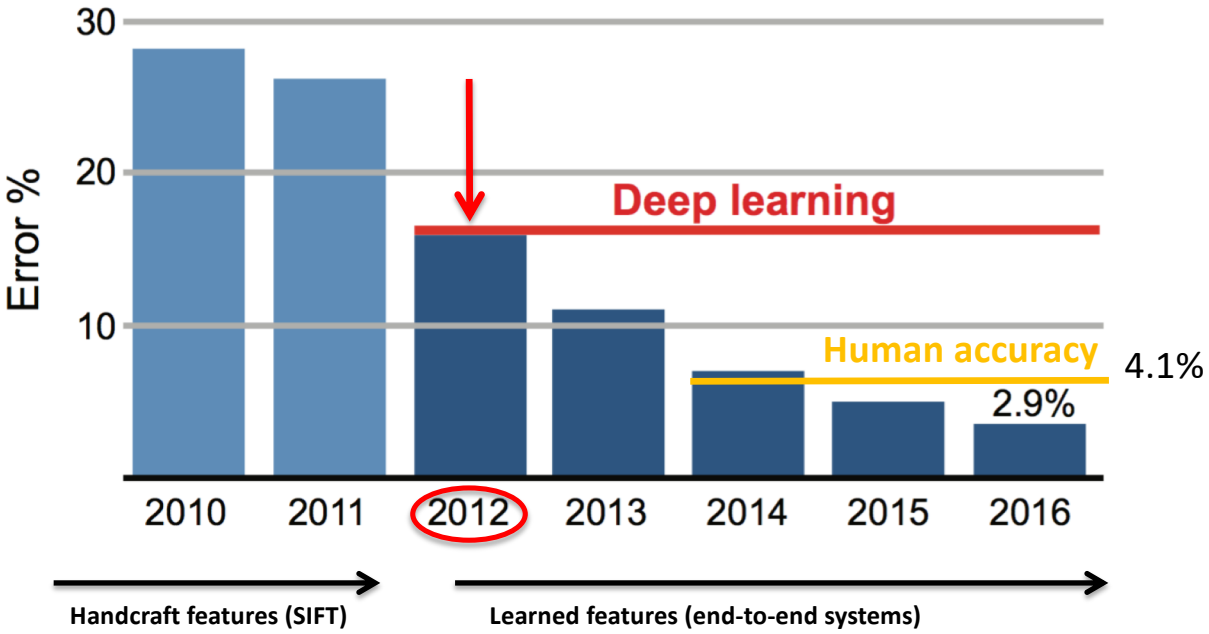
  - Architecture
  - Graph Structured Data

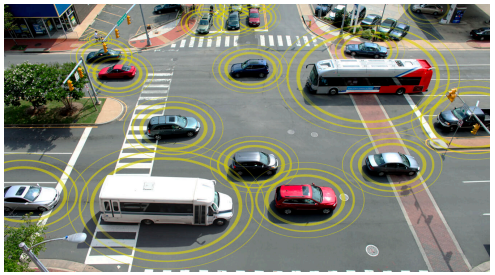
ConvNets

- A breakthrough in image recognition :

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

IMAGENET



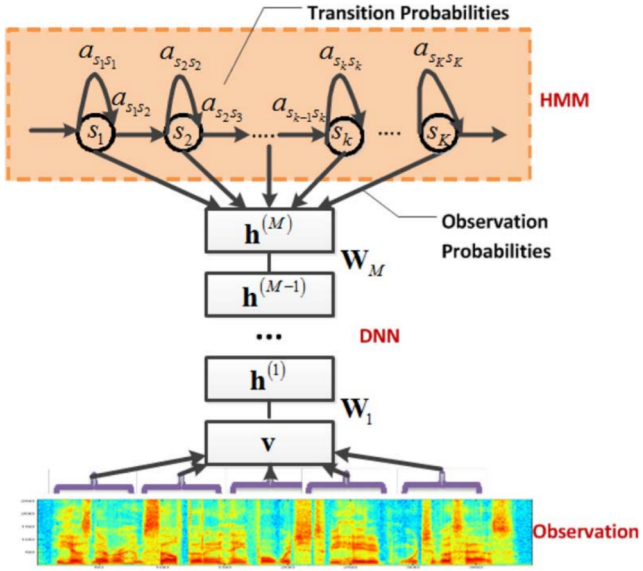


ConvNets

- A breakthrough in speech recognition :

  Dahl, Yu, Deng, Acero, 2010
  Hinton, Deng, Yu, Dahl et al. 2012

Acoustic model	Recog \ WER	RT03S FSH	Hub5 SWB
Traditional features	1-pass –adapt	27.4	23.6
Deep Learning	1-pass –adapt	18.5	16.1



ConvNets

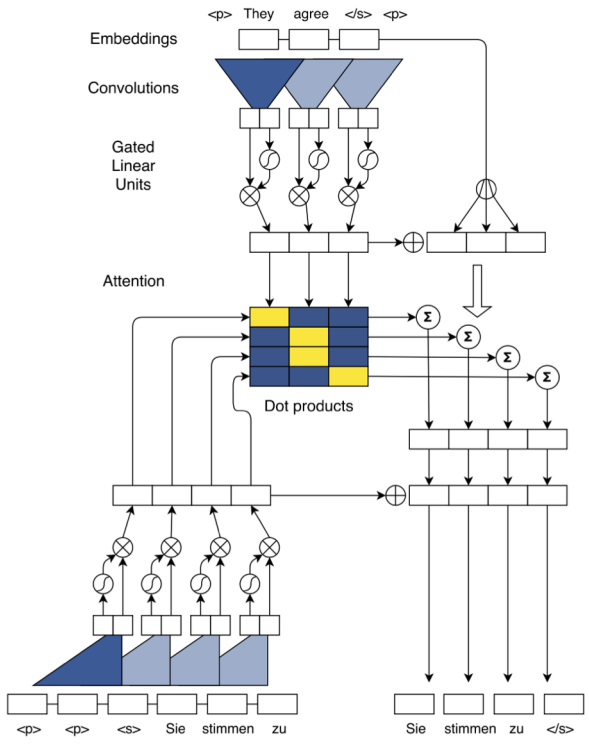
- A breakthrough in Machine Translation :

  Gehring, Auli, Grangier, Yarats, Dauphin, 2017

WMT'14 English-German	BLEU
Wu et al. (2016) GNMT	26.20
Wu et al. (2016) GNMT + RL	26.30
ConvS2S	26.43

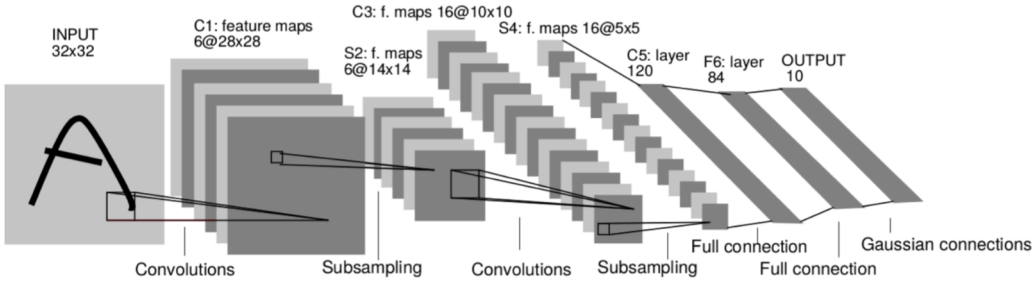
WMT'14 English-French	BLEU
Zhou et al. (2016)	40.4
Wu et al. (2016) GNMT	40.35
Wu et al. (2016) GNMT + RL	41.16
ConvS2S	41.44
ConvS2S (10 models)	41.62

	BLEU	Time (s)
GNMT GPU (K80)	31.20	3,028
GNMT CPU 88 cores	31.20	1,322
GNMT TPU	31.21	384
ConvS2S GPU (K40) $b = 1$	33.45	327
ConvS2S GPU (M40) $b = 1$	33.45	221
ConvS2S GPU (GTX-1080ti) $b = 1$	33.45	142
ConvS2S CPU 48 cores $b = 1$	33.45	142
ConvS2S GPU (K40) $b = 5$	34.10	587
ConvS2S CPU 48 cores $b = 5$	34.10	482
ConvS2S GPU (M40) $b = 5$	34.10	406
ConvS2S GPU (GTX-1080ti) $b = 5$	34.10	256



ConvNets

- An architecture for high-dimensional learning :

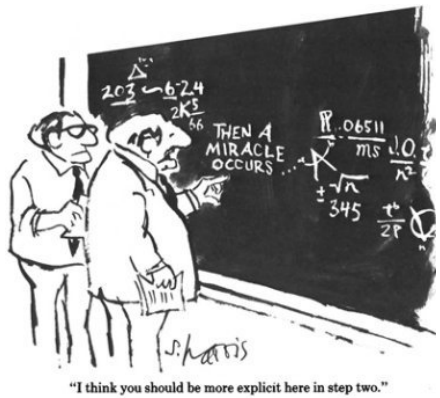


- Curse of dimensionality :

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

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

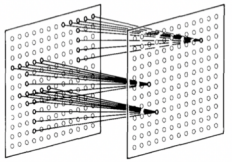
- ConvNets are powerful to solve high-dimensional learning problems.

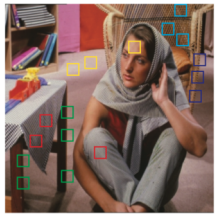


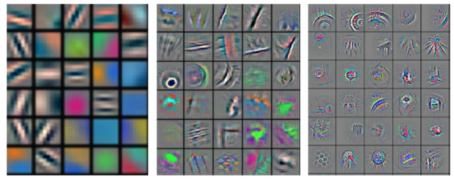
"I think you should be more explicit here in step two."

ConvNets

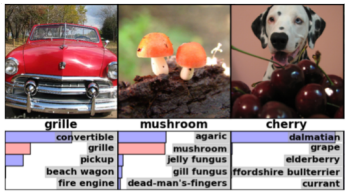
- Main assumption :
  - Data (image, video, sound) is compositional, it is formed of patterns that are:
    - Local (Hubel-Wiesel 1962)
    - Stationary (shared patterns)
    - Hierarchical (multi-scale)



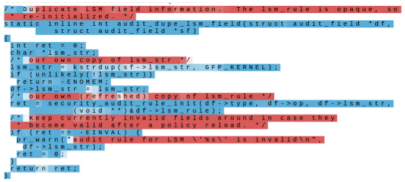




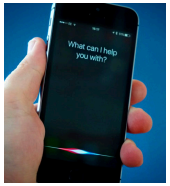
- ConvNets leverage the compositionality structure :
  - They extract compositional features and feed them to classifier, recommender, etc (end-to-end).



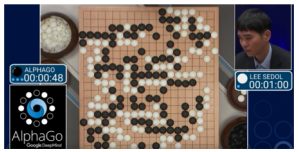
Computer Vision



NLP



Speech

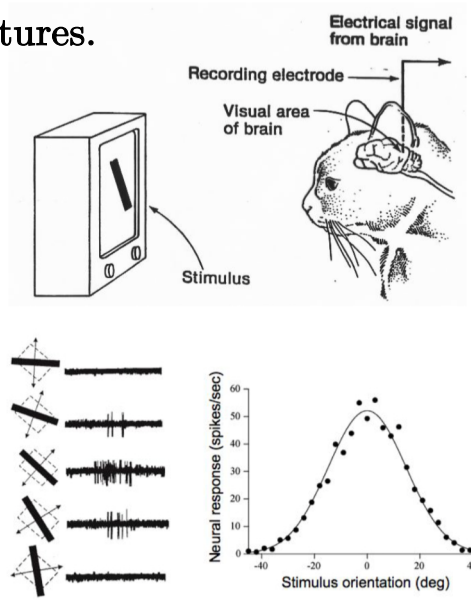


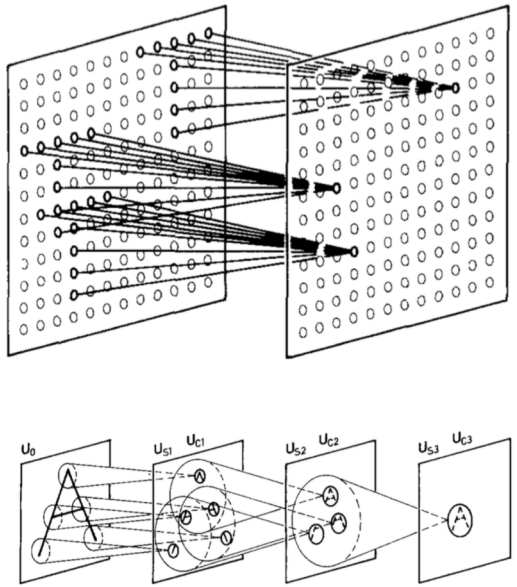
Games

Key Properties

- Locality :
  - Property inspired by the human visual cortex system.

- Local receptive fields (Hubel, Wiesel 1962) :
  - Activate in the presence of local features.

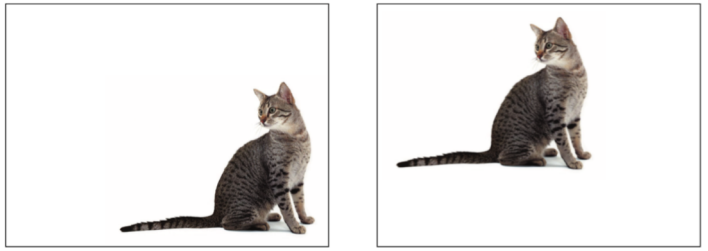




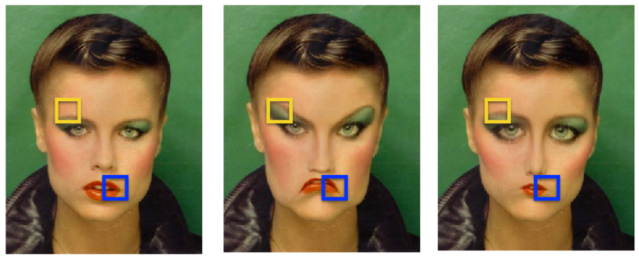
Neocognitron
Fukushima 1980

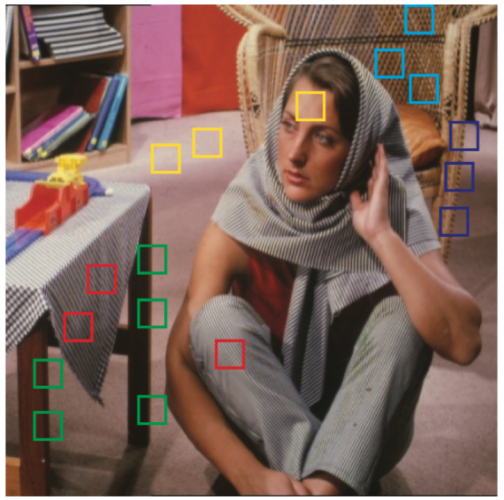
Key Properties

- Stationarity ⇔ Translation invariance
  - Global invariance



- Local stationarity ⇔ Similar patches are shared across the data domain
  - Local invariance, essential for intra-class variations

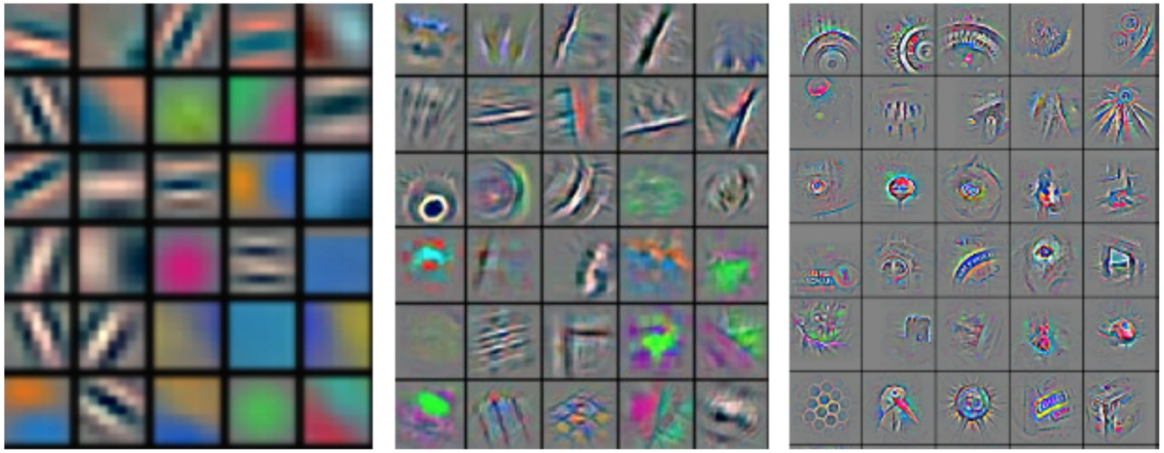




Key Properties

- Multi-scale :
  - Simple structures combine to compose slightly more abstract structures, and so on, in a hierarchical way.

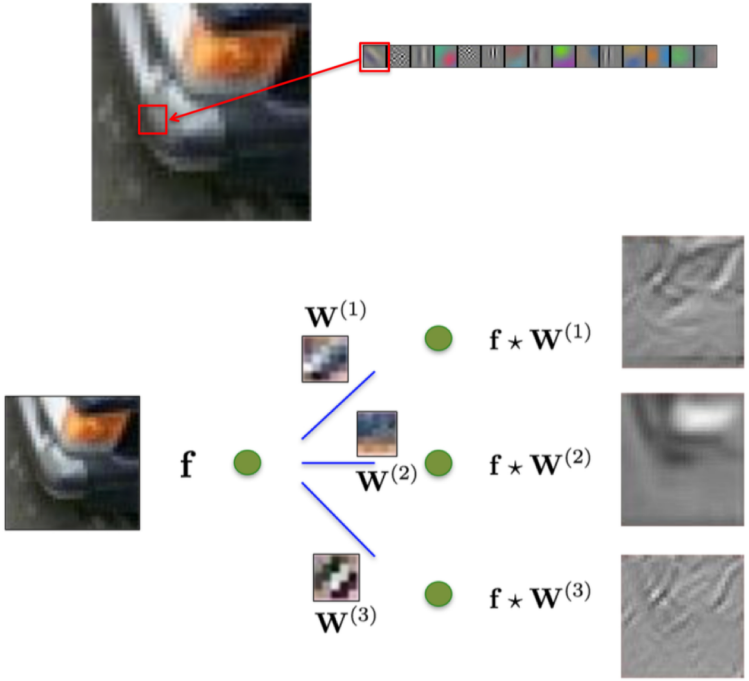
- Inspired by brain visual primary cortex (V1 and V2 neurons).

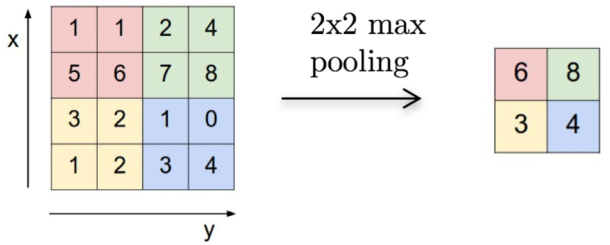


Features learned by ConvNet become increasingly more complex at deeper layers
(Zeiler, Fergus 2013)

Learning Complexity

- Locality :
  - Compact support kernels $\Rightarrow$ O(1) parameters per filter.
- Stationarity :
  - Convolutional operators $\Rightarrow$ O(nlogn) in general (FFT) and O(n) for compact kernels.
- Multi-scale :
  - Downsampling + pooling $\Rightarrow$ O(n)

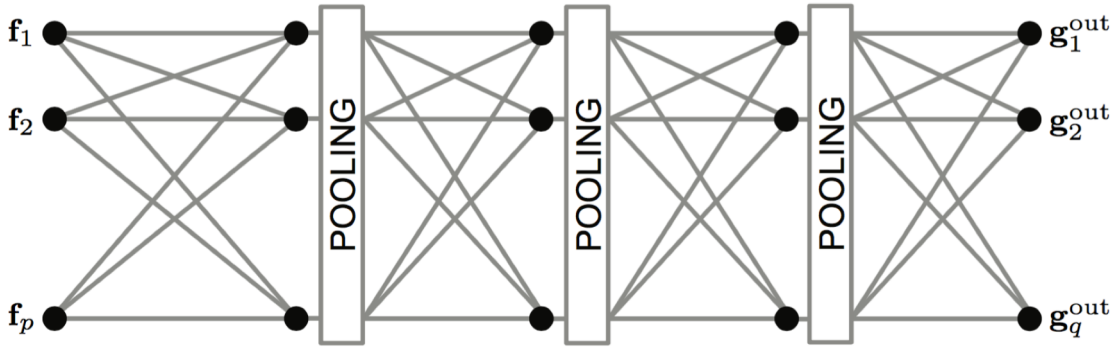




Compositional layers

$\mathbf{f}_l$ = l-th image feature (R,G,B channels), $\dim(\mathbf{f}_l) = n \times 1$

$\mathbf{g}_l^{(k)}$ = l-th feature map, $\dim(\mathbf{g}_l^{(k)}) = n_l^{(k)} \times 1$



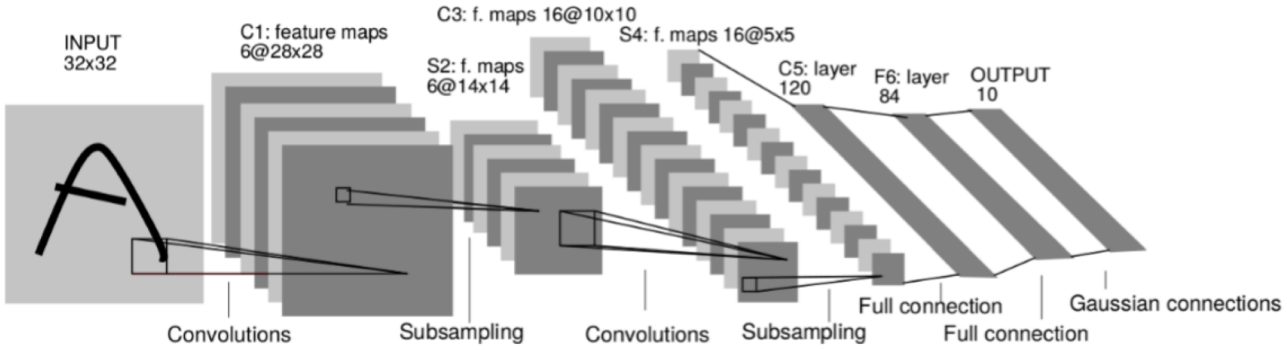
Compositional features consist of multiple convolutional + pooling layers.

Convolutional layer $$\mathbf{g}_l^{(k)} = \xi\left(\sum_{l'=1}^{q_{k-1}} \mathbf{W}_{l,l'}^{(k)} \star \xi\left(\sum_{l'=1}^{q_{k-2}} \mathbf{W}_{l,l'}^{(k-1)} \star \xi\left(\cdots \mathbf{f}_{l'}\right)\right)\right)$$

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

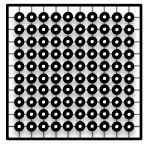
Pooling $\mathbf{g}_l^{(k)}(x) = \|\mathbf{g}_l^{(k-1)}(x') : x' \in \mathcal{N}(x)\|_p \quad p = 1, 2, \text{ or } \infty$

ConvNets



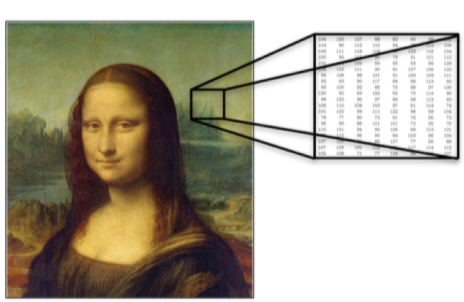
- ☺ Filters localized in space (locality)
- ☺ Convolutional filters (stationarity)
- ☺ Multiple layers (multi-scale)
- ☺ $O(1)$ parameters per filter (independent of input image size n)
- ☺ $O(n)$ complexity per layer (filtering done in the spatial domain)

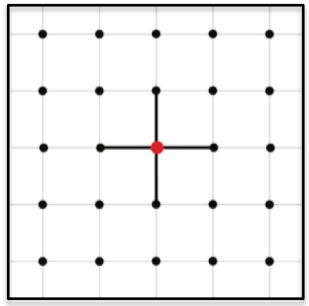
Outline

- Part 1: Traditional ConvNets 
  - Architecture
  - Graph Structured Data

Data Domain

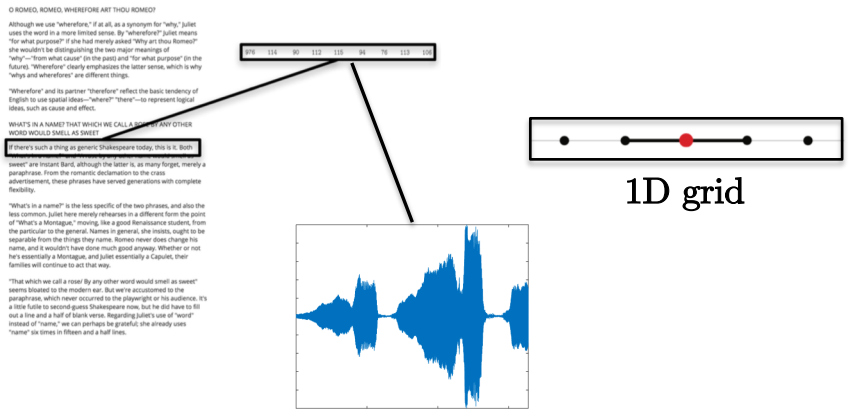
- Image, volume, video lie on
  2D, 3D, 2D+1 Euclidean domains





2D grids

- Sentence, word, sound lie on
  1D Euclidean domain

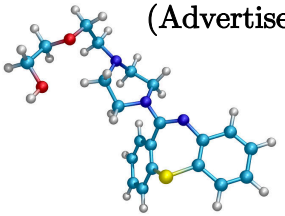


1D grid

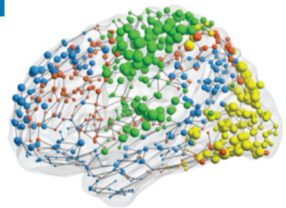
- These domains have strong regular spatial structures.
  - All ConvNet operations are mathematically well defined and fast (convolution, pooling).

Graph Structured Data

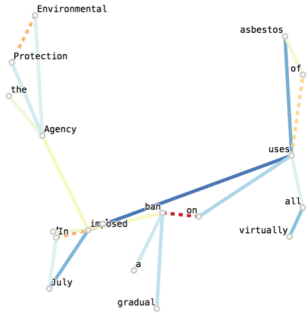
Social networks (Advertisement)



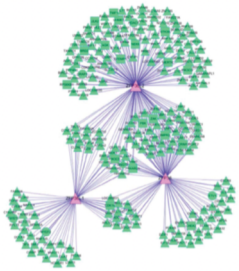
Drug/Material molecules (Chemistry)



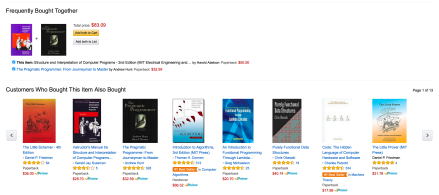
Brain connectivity (Neuroscience)



Words relationships (NLP)

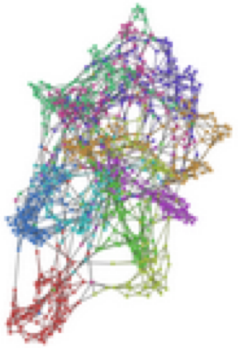


Gene Regulatory Network

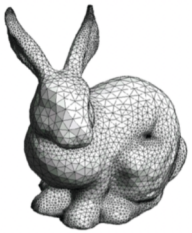


Recommender systems (Amazon, Netflix)

=

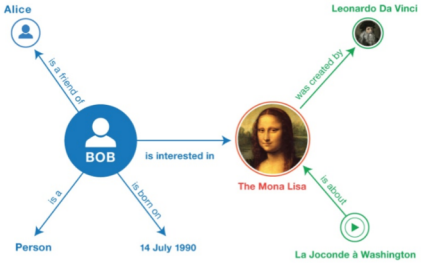


Graphs/ Networks

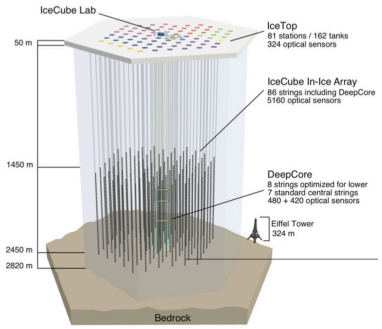


3D Meshes (Computer Graphics)

Transportation networks



Knowledge graph (Causality)



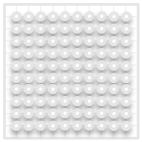
Neutrino detection (High-energy Physics)

New Challenges

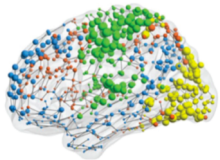
- How to extend ConvNets to graph structured data ?
- Assumption :
  - Graph structured data is compositional :
    - Locally stationary and manifest hierarchical structures.
- How to define compositionality on graphs ? (convolution/pooling on graphs)
- How to make them fast? (linear complexity)

Outline

- Part 1: Traditional ConvNets



- **Part 2: Spectral Graph ConvNets**

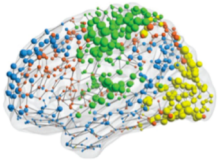


- Part 3: Spatial Graph ConvNets

- Conclusion

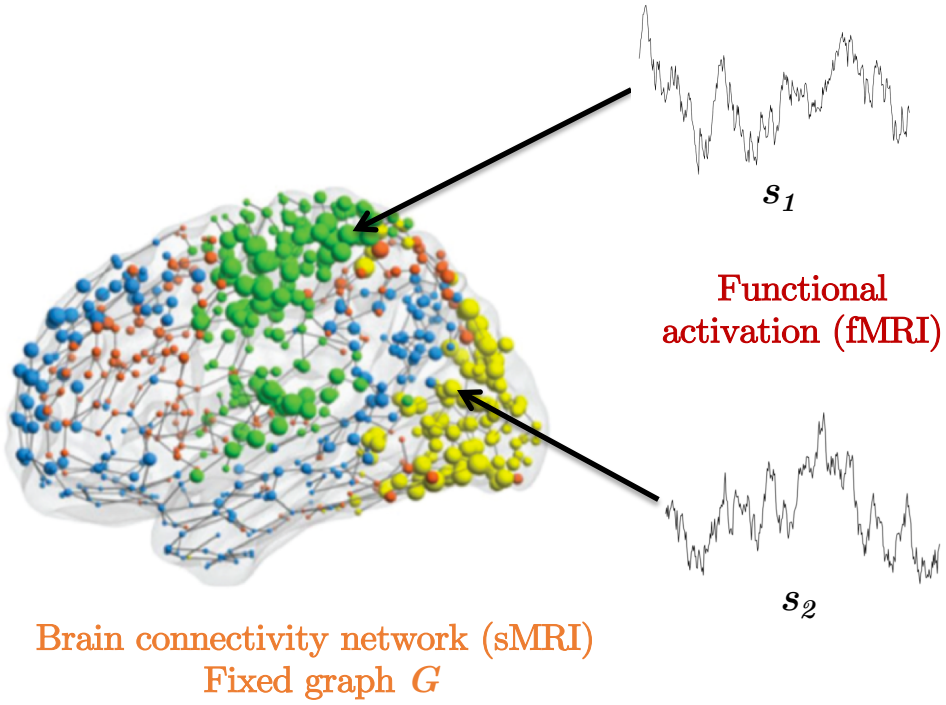
Outline

- Part 2: Spectral Graph ConvNets
  - Problem Setting
  - Graph Theory
    - Spectral Graph Convolution
    - Graph Pooling
  - Spectral GCNs
  - Applications

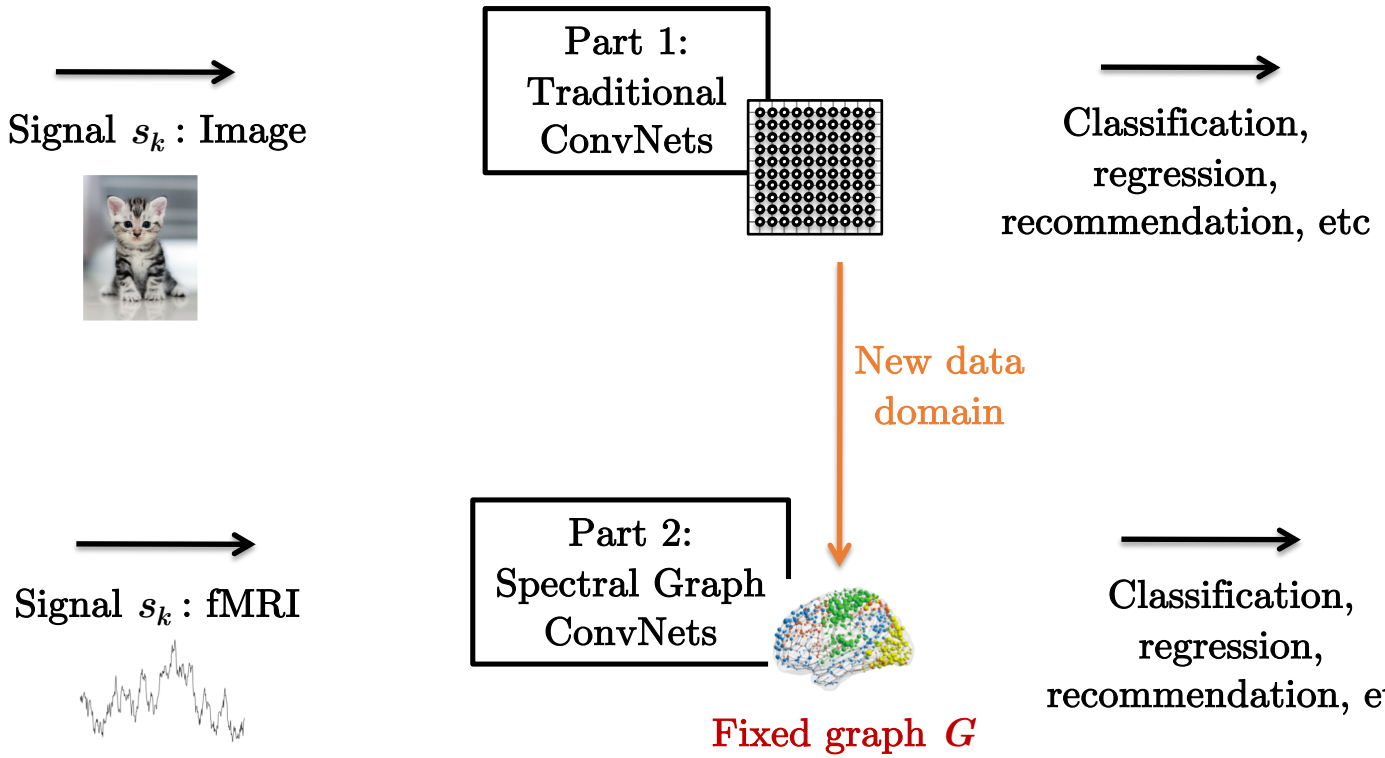


Problem Setting

- Given fixed arbitrary graph(s) G, and a set of signals s_k on G to be analyzed :



Spectral ConvNets



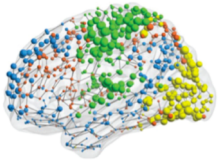
Generalization challenges

- Convolution and down-sampling (for pooling) must be generalized from Euclidean grid domains to graphs.

- How ?
  - Spectral graph theory allows to redefine convolution in the context of graphs with Fourier analysis.
  - Graph theory provide graph clustering techniques to reformulate down-sampling for graphs.

Outline

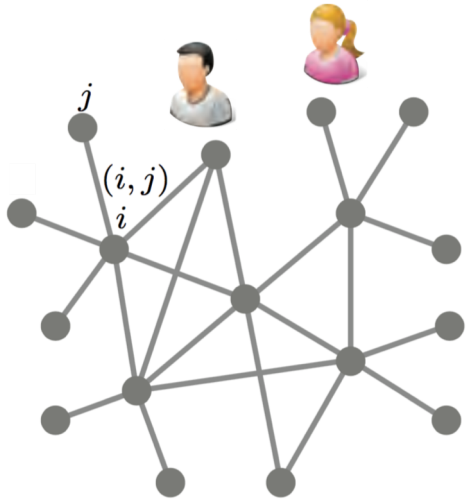
- Part 2: Spectral Graph ConvNets
  - Problem Setting
  - Graph Theory
    - Spectral Graph Convolution
    - Graph Pooling
  - Spectral GCNs
  - Applications



Graphs

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

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



Graph Laplacian

- Laplacian operator $\Delta : L^2(\mathcal{V}) \to L^2(\mathcal{V})$

$$(\Delta f)_i = \frac{1}{b_i} \sum_{j:(i,j)\in\mathcal{E}} a_{ij}(f_i - f_j)$$

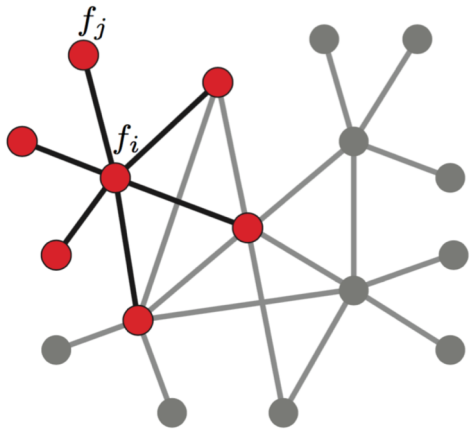
  difference between f and its local average (2^{nd} derivative on graphs)

- Core operator in spectral graph theory.

- Represented as a positive semi-definite $n \times n$ matrix

  - Un-normalized Laplacian $\mathbf{\Delta} = \mathbf{D} - \mathbf{A}$
  - Normalized Laplacian $\mathbf{\Delta} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$
  - Random walk Laplacian $\mathbf{\Delta} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$

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



Spectral Decomposition

- A Laplacian of a graph of n vertices admits n eigenvectors:

$$\boldsymbol{\Delta}\boldsymbol{\phi}_k = \lambda_k \boldsymbol{\phi}_k, \qquad k = 1, 2, \ldots$$

  - Eigenvectors are real and orthonormal (self-adjointness) $\quad \langle \phi_k, \phi_{k'} \rangle_{L^2(\mathcal{V})} = \delta_{kk'}$
  - Eigenvalues are non-negative (positive-semidefiniteness) $\quad 0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$
  - Laplacian eignenvectors are also called Fourier basis functions/modes.

- Eigen-decomposition of graph Laplacian:

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

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

Interpretation

- Find the smoothest orthonormal basis $\mathbf{\Phi} = (\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_n)$ on a graph

$$\min_{\boldsymbol{\phi}_k} E_{\text{Dir}}(\boldsymbol{\phi}_k) \quad \text{s.t.} \quad \|\boldsymbol{\phi}_k\| = 1, \quad k = 2, 3, \dots n$$
$$\boldsymbol{\phi}_k \perp \text{span}\{\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_{k-1}\}$$

  where E_{Dir} is the Dirichlet energy = measure of smoothness of a function

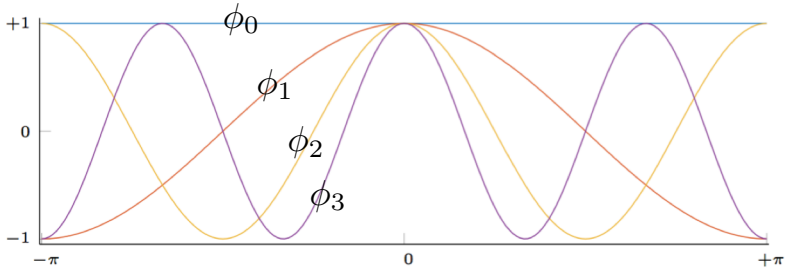
$$E_{\text{Dir}}(\mathbf{f}) = \mathbf{f}^\top \mathbf{\Delta} \mathbf{f}$$

- Solution :
  - Smallest n Laplacian eigenvectors :

$$\min_{\mathbf{\Phi} \in \mathbb{R}^{n \times n}} \underbrace{\text{trace}(\mathbf{\Phi}^\top \mathbf{\Delta} \mathbf{\Phi})}_{\|\mathbf{\Phi}\|_{\mathcal{G}} \text{ Dirichlet norm}} \quad \text{s.t.} \quad \mathbf{\Phi}^\top \mathbf{\Phi} = \mathbf{I}$$

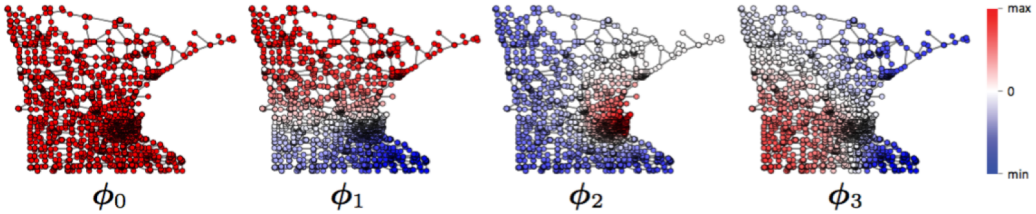
Fourier Modes

- Euclidean domain :



First eigenvectors of 1D Euclidean Laplacian = standard Fourier basis

- Graph domain :



First Laplacian eigenvectors of a graph

Laplacian eigenvectors related to graph geometry!

(s.a. communities, hubs, etc)

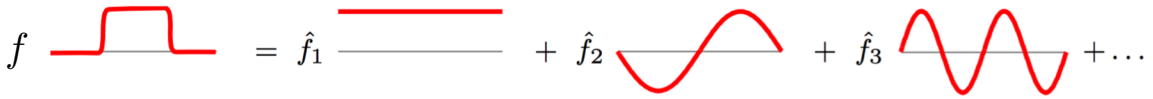
Spectral Graph Clustering[1]

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

Euclidean Fourier analysis

- **A function** $f : [-\pi, \pi] \to \mathbb{R}$ **can be written as** Fourier series **:**

$$f(x) = \sum_{k \geq 0} \underbrace{\frac{1}{2\pi} \int_{-\pi}^{\pi} f(x') e^{-ikx'} dx'}_{\hat{f}_k = \langle f, e^{-ikx} \rangle_{L^2([-\pi,\pi])}} e^{-ikx}$$



- **Fourier basis** e^{-ikx} $=$ Laplace-Beltrami eigenfunctions **:**

$$-\Delta \phi_k = k^2 \phi_k$$

$$\begin{cases} \phi_k & = & \text{Fourier mode} \\ k & = & \text{frequency of Fourier mode} \end{cases}$$

Fourier analysis on graphs

- A function $f : \mathcal{V} \to \mathbb{R}$ can be written as Fourier series[1]:

$$f_i = \sum_{k=1}^{n} \underbrace{\langle f, \phi_k \rangle_{L^2(\mathcal{V})}}_{\hat{f}_k} \phi_{k,i}$$

- $\hat{f}_k$ is the k-th graph Fourier coefficient.
- In matrix-vector notation, with the n x n Fourier matrix $\mathbf{\Phi} = [\phi_1, ..., \phi_n]$

$$\hat{\mathbf{f}} = \mathbf{\Phi}^\top \mathbf{f} \quad \text{and} \quad \mathbf{f} = \mathbf{\Phi}\hat{\mathbf{f}}$$

Fourier transform

Inverse Fourier transform

- Graph Fourier basis ϕ_k = Laplacian eigenvectors :

$$\mathbf{\Delta}\phi_k = \lambda_k \phi_k \quad \text{with} \quad \begin{cases} \phi_k & = & \text{graph Fourier mode} \\ \lambda_k & = & \text{(square) frequency} \end{cases}$$

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

Convolution in Continuous Euclidean Space

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

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

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

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

  - Efficient computation :
    - FFT complexity is $O(n \log n)$

Convolution in Discrete Euclidean Space

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

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

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

diagonalised by Fourier basis (Toeplitz)

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

Convolution on Graphs

- Spectral convolution of $f, g \in L^2(\mathcal{V})$ can be defined by analogy[1]:

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

- In matrix-vector notation

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

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

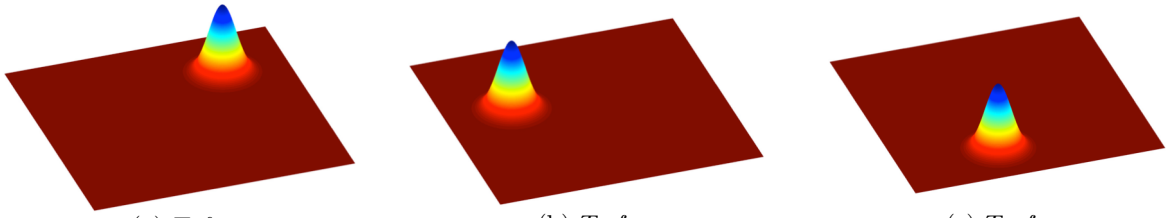
[1] K. Hammond, P. Vandergheynst, R. Gribonval, Wavelets on graphs via spectral graph theory, 2011

No Shift Invariance on Graphs

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

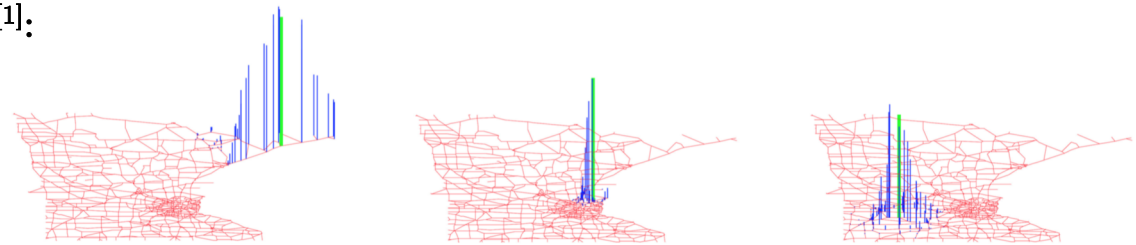
$$T_i \mathbf{f} = \mathbf{f} \star \boldsymbol{\delta}_i$$

- Euclidean domain :



(a) $T_s f$ (b) $T_{s'} f$ (c) $T_{s''} f$

- Graph domain[1]:

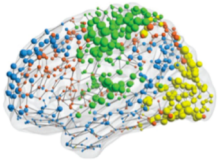


(d) $T_i f$ (e) $T_{i'} f$ (f) $T_{i''} f$

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

Outline

- Part 2: Spectral Graph ConvNets
  - Problem Setting
  - Graph Theory
    - Spectral Graph Convolution
    - Graph Pooling
  - Spectral GCNs
  - Applications



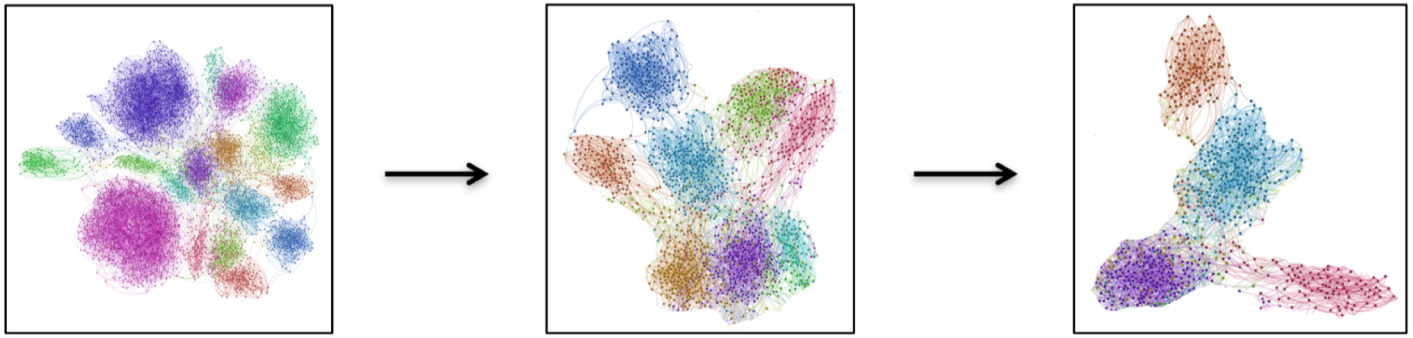
Graph Down-Sampling

- Goals :
  - Pool similar local features with max/average pooling.
    - Create invariance to local geometric deformations.
  - Series of pooling layers create invariance to global geometric deformations.

- Challenges :
  - Design a multi-scale coarsening algorithm that preserves non-linear graph structures.
  - How to make graph pooling fast?

Graph Down-Sampling

- Graph down-sampling ⇔ graph coarsening ⇔ graph partitioning :
  - Decompose G into meaningful clusters.



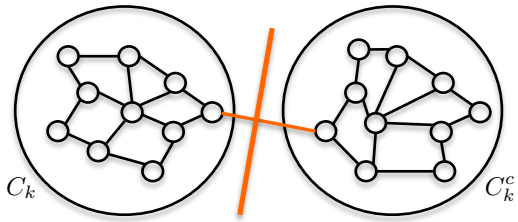
$G^{l=0} = G$ $G^{l=1}$ $G^{l=2}$

- Graph partitioning is NP-hard ⇒ Approximation

Balanced Cuts

- Powerful combinatorial graph partitioning models :
  - Normalized cut (Shi, Malik, 2000)

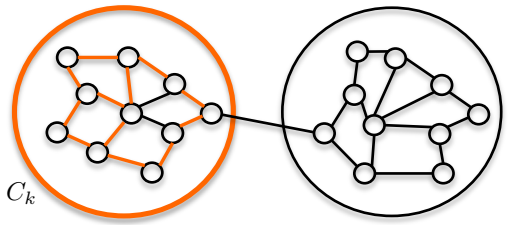
$$\min_{C_1,\dots,C_K} \sum_{k=1}^{K} \frac{\mathrm{Cut}(C_k, C_k^c)}{\mathrm{Vol}(C_k)}$$



Partitioning by min edge cuts.

  - Normalized association

$$\max_{C_1,\dots,C_K} \sum_{k=1}^{K} \frac{\mathrm{Assoc}(C_k)}{\mathrm{Vol}(C_k)}$$



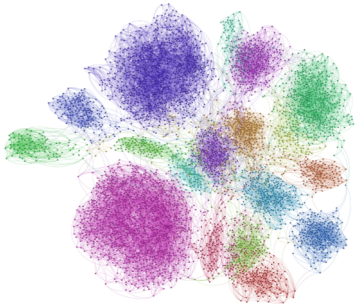
Partitioning by max vertex matching.

where $\mathrm{Cut}(A,B) := \sum_{i\in A, j\in B} a_{ij}$, $\mathrm{Assoc}(A) := \sum_{i\in A, i\in B} a_{ij}$,
$\mathrm{Vol}(A) := \sum_{i\in A, j\in B} d_i$, and $d_i := \sum_{j\in V} a_{ij}$.

- Both models are equivalent, but lead to different algorithms.

Balanced Cuts

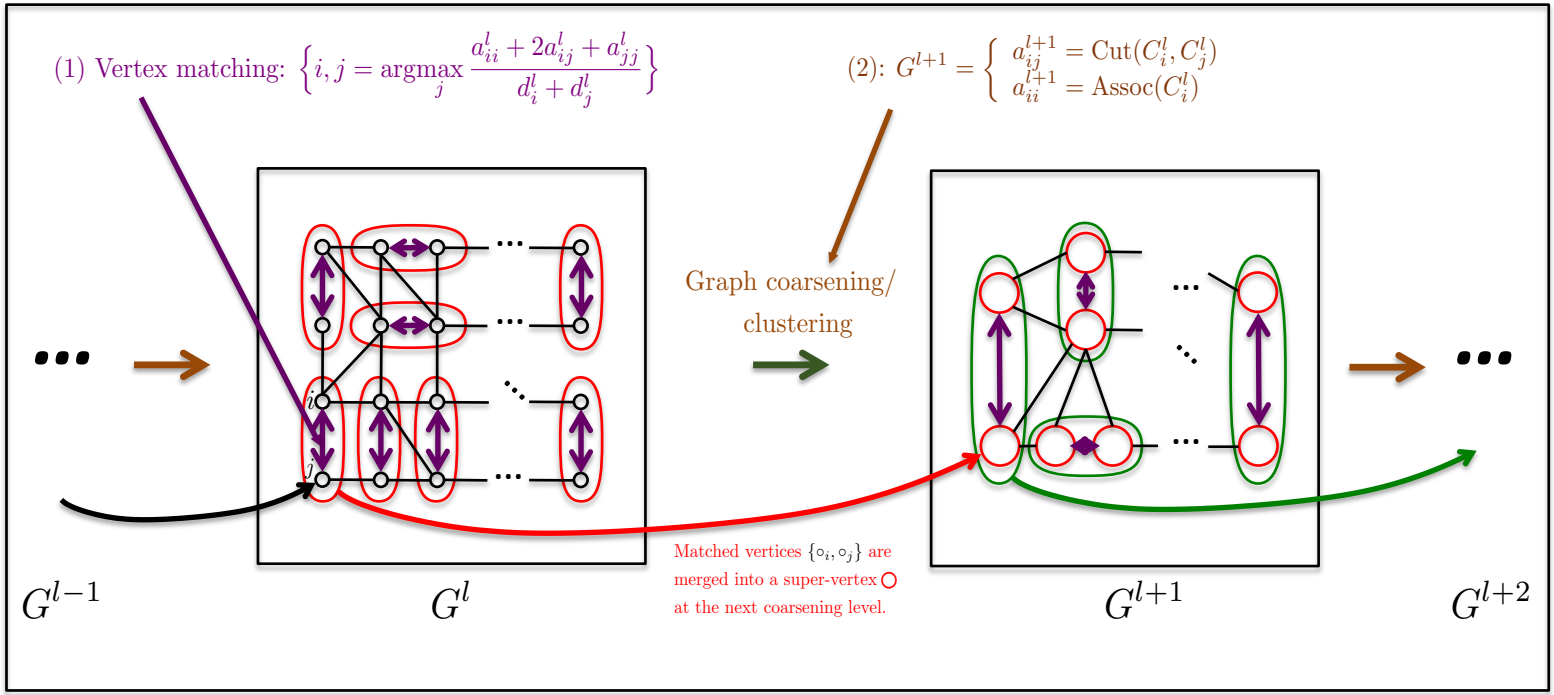
- Balanced cuts are NP-hard :
  - Most popular approximation techniques focus on linear spectral relaxation (eigenproblem with global solution).
- Graph geometry are generally not linear :
  - Graclus (Dhillon, Guan, Kulis 2007) algorithm computes non-linear clusters that locally maximize the Normalized Association.



- Graclus algorithm offers a control of the coarsening ratio of ≈ 2 (like image grid) using heavy-edge matching (HEM) by Karypis, Kumar 1995.

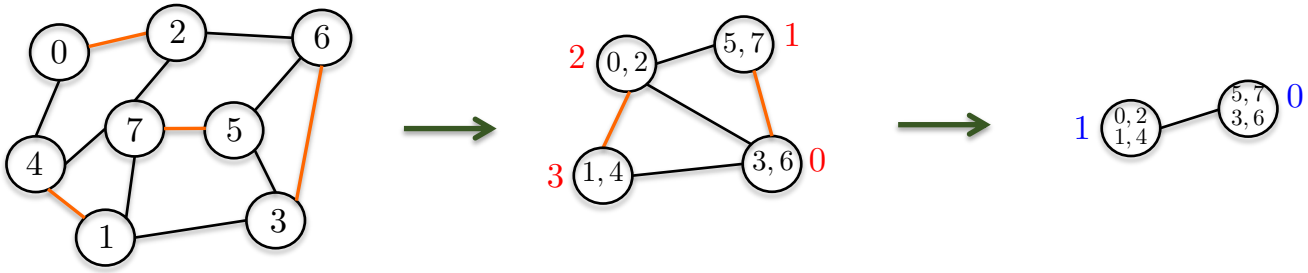
Heavy-Edge Matching (HEM)

- HEM proceeds by two successive steps (that guarantees a local solution of Norm Assoc):
  - Vertex matching
  - Graph coarsening



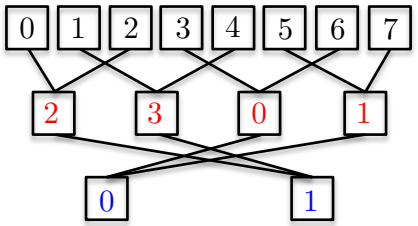
Unstructured Pooling

- Sequence of coarsened graphs produced by HEM :



$G^{l=0} = G$ $G^{l=1}$ $G^{l=2}$

Coarsening structure

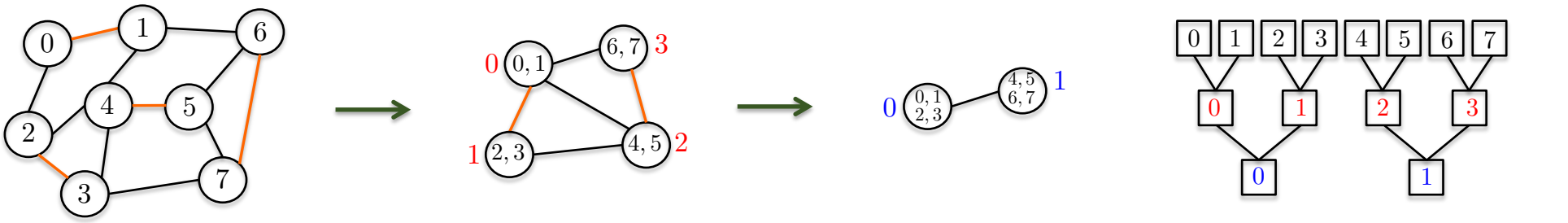


Unstructured arrangement of nodes

- ☹ Stores a table of indices for graph and all its coarsened versions.
- ☹ Computationally inefficient

Fast Graph Pooling[1]

- Structured pooling :
  - Arrangement of the node indexing such that adjacent nodes are hierarchically merged at the next coarser level.



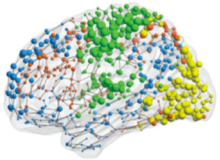
$G^{l=0} = G$ $G^{l=1}$ $G^{l=2}$ Binary tree arrangement of nodes

☺ As efficient as 1D-Euclidean grid pooling !

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

Outline

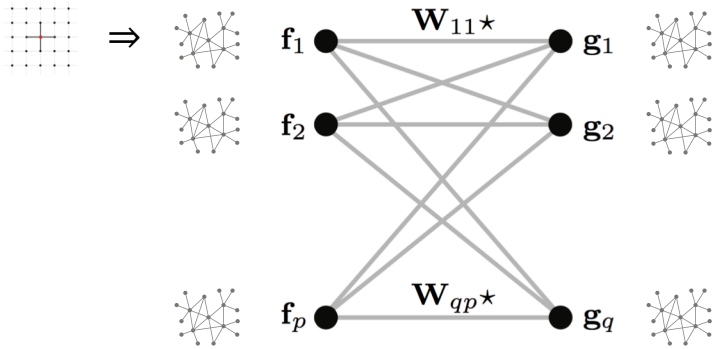
- Part 2: Spectral Graph ConvNets
  - Problem Setting
  - Graph Theory
    - Spectral Graph Convolution
    - Graph Pooling
  - Spectral GCNs
  - Applications



Vanilla Spectral GCNs[1]

- Graph convolutional layer :

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



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

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

[1] Bruna, Zaremba, Szlam, LeCun 2014

Spectral Graph Convolution

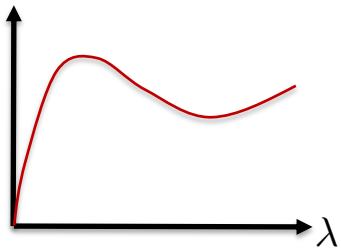
- Convolutional layer in the spatial domain:

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

where $\mathbf{W}_{l,l'}$ = matrix of graph spatial filter,

can also be expressed in the spectral domain (using $\mathbf{g} \star \mathbf{f} = \mathbf{\Phi}\, \hat{g}(\mathbf{\Lambda})\mathbf{\Phi}^\top \mathbf{f}$)

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



where $\hat{\mathbf{W}}_{l,l'}$ = n x n diagonal matrix of graph spectral filter.

We will denote the spectral filter without the hat symbol, i.e. $\mathbf{W}_{l,l'}$

Vanilla Spectral GCNs[1]

- Series of spectral convolutional layers :

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

with spectral coefficients $\mathbf{W}_{l,l'}^{(k)}$ to be learned at each layer.

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

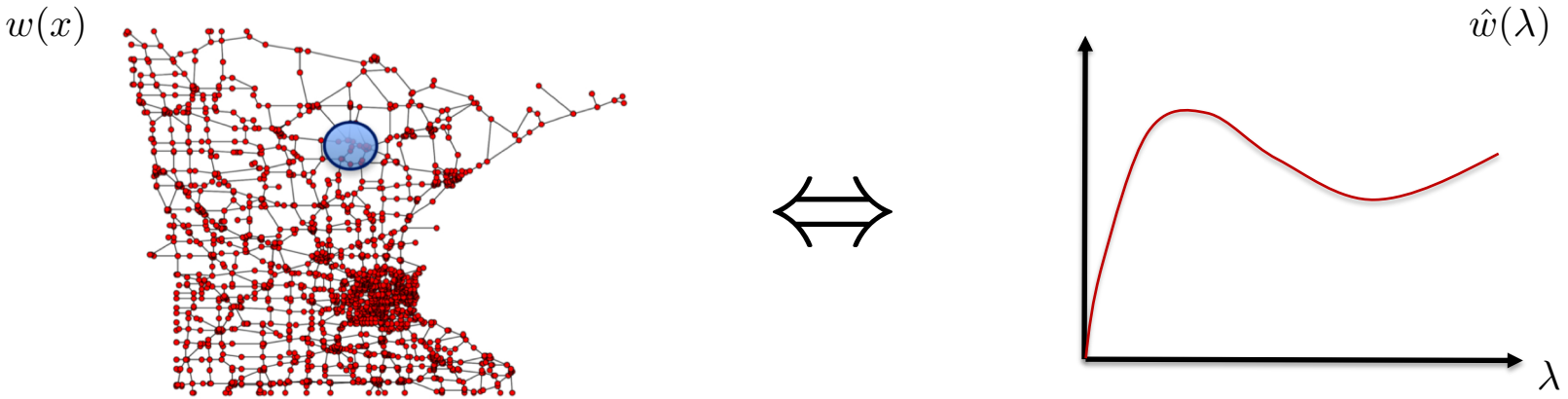
[1] Bruna, Zaremba, Szlam, LeCun 2014

Spatial Localization[1]

- Parseval's identity

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

$\Rightarrow$ Localization in space = smoothness in frequency domain



Spatial localization

Smooth spectral filter

[1] Henaff, Bruna, LeCun 2015

Smooth Spectral Filter

- Parametrize a smooth spectral filter function $w(\lambda)$ with a linear combination of smooth kernel functions $\beta_1(\lambda), \ldots, \beta_r(\lambda)$, here splines[1]:

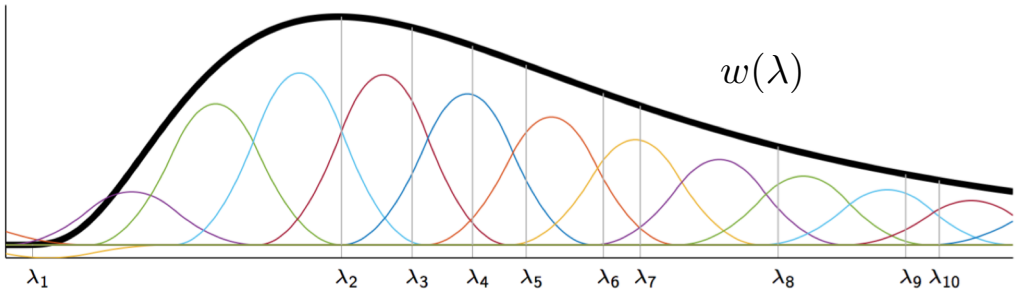
$$w_{\boldsymbol{\alpha}}(\lambda) = \sum_{j=1}^{r} \alpha_j \beta_j(\lambda)$$

$$\Downarrow$$

$$w_{\boldsymbol{\alpha}}(\lambda_i) = \sum_{j=1}^{r} \alpha_j \beta_j(\lambda_i) = (\mathbf{B}\boldsymbol{\alpha})_i \quad \Longrightarrow \quad \mathbf{W} = \mathrm{Diag}(\mathbf{B}\boldsymbol{\alpha})$$

Splines

Coefficients (learned by backpropagation)



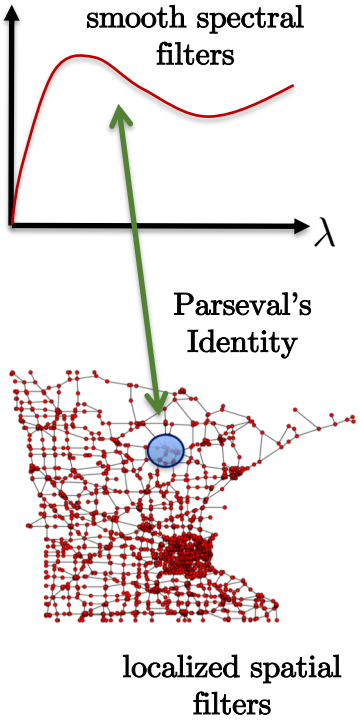
[1] Henaff, Bruna, LeCun 2015

SplineGCNs[1,2]

- Series of graph convolutional layers:

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

with smooth spectral filters at each layer.



- ☺ Fast-decaying filters in space
- ☺ O(1) parameters per layer
- ☹ O(n^2) computation of forward and inverse Fourier transforms ϕ, $\phi^\top$ (no FFT on graphs)
- ☹ Filters are basis-dependent ⇒ does not generalize across graphs

[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

Polynomial Spectral Filters[1]

- Represent spectral smooth functions with polynomials of Laplacians :

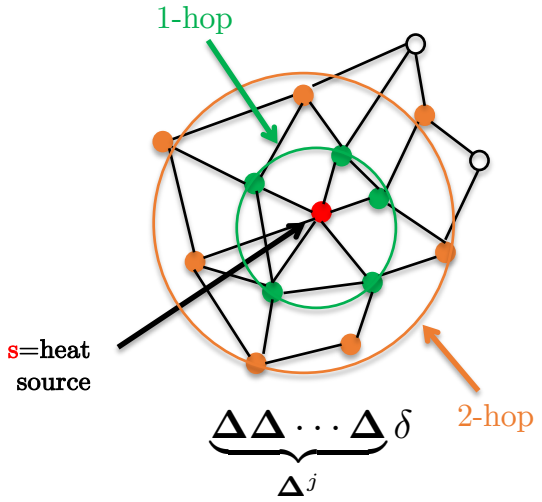
$$w_{\boldsymbol{\alpha}}(\lambda) = \sum_{j=0}^{r} \alpha_j \lambda^j$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_r)^\top$ is the vector of filter parameters.

- Convolutional layer : Apply spectral filter to feature signal f

$$w_{\boldsymbol{\alpha}}(\boldsymbol{\Delta})\mathbf{f} = \textstyle\sum_{j=0}^{r} \alpha_j \boldsymbol{\Delta}^j \mathbf{f}$$

- Key observation: Each Laplacian operation increases the support of a function by 1-hop ⇒ Exact control the size of Laplacian-based filters.



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

Linear Complexity

- Application of the filter to a feature signal f :

$$w_{\boldsymbol{\alpha}}(\boldsymbol{\Delta})\mathbf{f} = \sum_{j=0}^{r} \alpha_j \boldsymbol{\Delta}^j \mathbf{f}$$

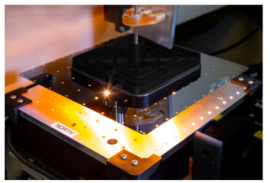
- Denote $\mathbf{X}_0 = \mathbf{f}$ and define $\mathbf{X}_1 = \boldsymbol{\Delta}\mathbf{X}_0 = \boldsymbol{\Delta}\mathbf{f}$ and the sequence $\mathbf{X}_j = \boldsymbol{\Delta}\mathbf{X}_{j-1}$

$$w_{\boldsymbol{\alpha}}(\boldsymbol{\Delta})\mathbf{f} = \sum_{j=0}^{r} \alpha_j \mathbf{X}_j$$

- Two important observations:

  1. *No* computation of the eigen-decomposition of Laplacian (ϕ,Λ) is required.
  2. Observe that the sequence $\{X_j\}$ is generated by multiplication of a sparse matrix Δ and a vector $\Rightarrow$ Complexity is O(E.r)=$O(n)$ for sparse (real-world) graphs.

- Graph convolutional layers are GPU friendly (but not yet optimized or soon to be).



cerebras Aug. 2019

PolyGCNs[1]

- Series of spectral convolutional layers with monomials of Laplacian $1, x, x^2, x^3, \cdots$:

$$w_{\boldsymbol{\alpha}}(\boldsymbol{\Delta})\mathbf{f} = \sum_{j=0}^{r} \alpha_j \boldsymbol{\Delta}^j \mathbf{f}$$

- Properties :

☺	Filters are exactly localized in r-hops support
☺	O(1) parameters per layer
☺	No eigen-decomposition/ϕ,$\phi^\intercal$ ⇒ O(n) computational complexity (assuming sparsely-connected graphs)
☹	Unstable under coefficients perturbation (hard to optimize)
☹	Filters are basis-dependent ⇒ does not generalize across graphs

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

Chebyshev Polynomials

- Graph convolution with non-orthogonal monomial basis $1, x, x^2, x^3, \cdots$

$$w_{\boldsymbol{\alpha}}(\boldsymbol{\Delta})\mathbf{f} = \sum_{j=0}^{r} \alpha_j \boldsymbol{\Delta}^j \mathbf{f}$$

- Graph convolution with orthogonal Chebyshev polynomials :

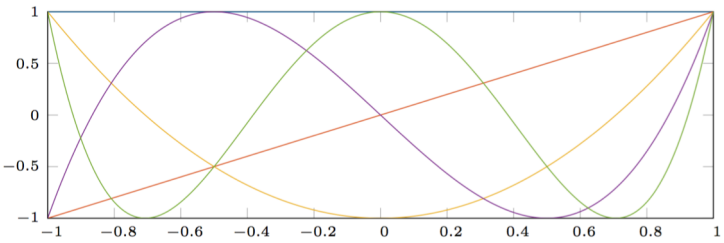
$$w_{\boldsymbol{\alpha}}(\tilde{\boldsymbol{\Delta}})\mathbf{f} = \sum_{j=0}^{r} \alpha_j T_j(\tilde{\boldsymbol{\Delta}})\mathbf{f}$$

$\tilde{\boldsymbol{\Delta}} = 2\lambda_n^{-1}\boldsymbol{\Delta} - \mathbf{I}$

Scaled Laplacian

- Orthonormal on $L^2([-1,+1])$ w.r.t. $\langle f, g \rangle = \int_{-1}^{+1} f(\tilde{\lambda}) g(\tilde{\lambda}) \frac{d\tilde{\lambda}}{\sqrt{1-\tilde{\lambda}^2}}$

- Stable under perturbation of coefficients



T_0, T_1, T_2, T_3

ChebGCNs[1]

- Application of the Chebyshev filter to a feature signal f :

with $$w_{\boldsymbol{\alpha}}(\tilde{\boldsymbol{\Delta}})\mathbf{f} = \sum_{j=0}^{r} \alpha_j T_j(\tilde{\boldsymbol{\Delta}})\mathbf{f} = \sum_{j=0}^{r} \alpha_j \mathbf{X}^{(j)}$$

$$\begin{aligned} \mathbf{X}^{(j)} &= T_j(\tilde{\boldsymbol{\Delta}})\mathbf{f} \\ &= 2\tilde{\boldsymbol{\Delta}}\mathbf{X}^{(j-1)} - \mathbf{X}^{(j-2)}, \qquad \mathbf{X}^{(0)} = \mathbf{f}, \quad \mathbf{X}^{(1)} = \tilde{\boldsymbol{\Delta}}\mathbf{f} \end{aligned}$$

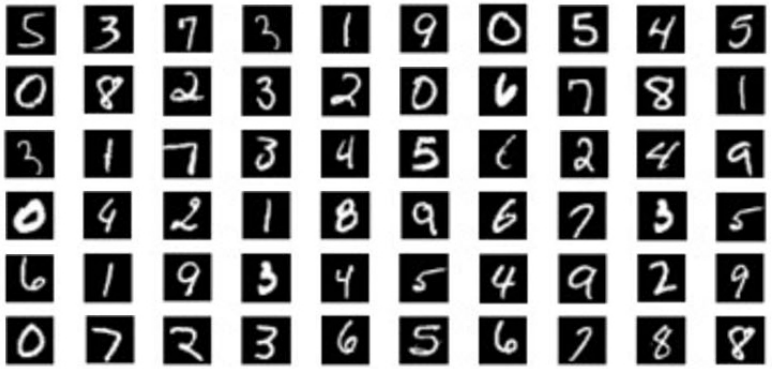
- ☺ Filters are exactly localized in r-hops support
- ☺ O(1) parameters per layer
- ☺ No eigen-decomposition/ϕ,$\phi^\intercal$ ⇒ O(n) computational complexity (assuming sparsely-connected graphs)
- ☺ Stable under coefficients perturbation
- ☹ Filters are basis-dependent ⇒ does not generalize across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

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

Numerical Experiments

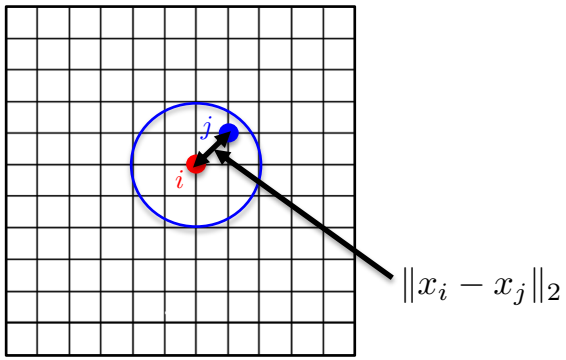
- ChebGCNs for Image Classification.

- MNIST : 70,000 images represented on a 2D grid of size 28x28 (dim=282=784) of handwritten digits, from 0 to 9.



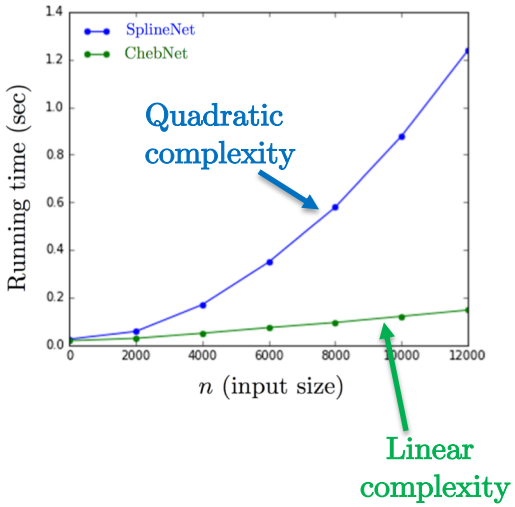
- Graph :
  - A k-NN graph (k=8) of the Euclidean grid :

$$W_{ij} = e^{-\|x_i - x_j\|_2^2/\sigma}$$



Numerical Experiments

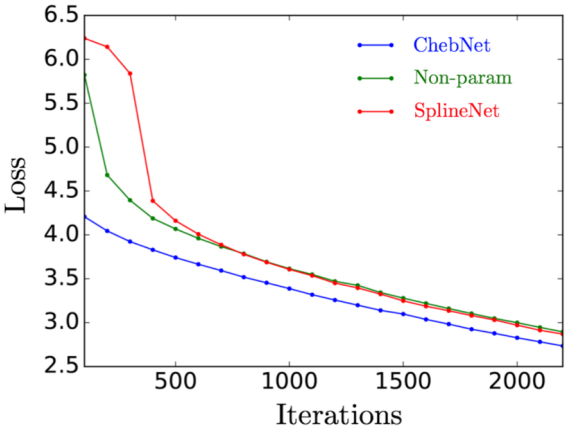
- Running time

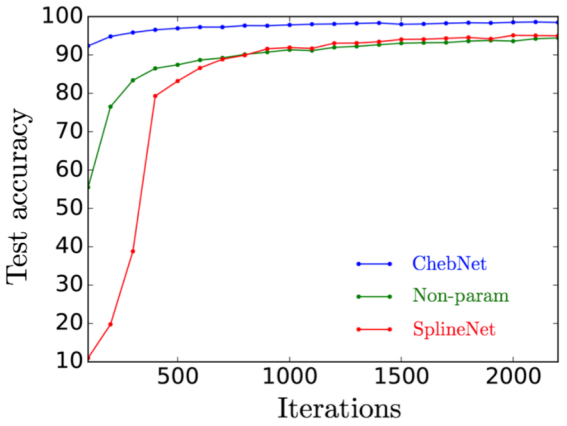


- Accuracy

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

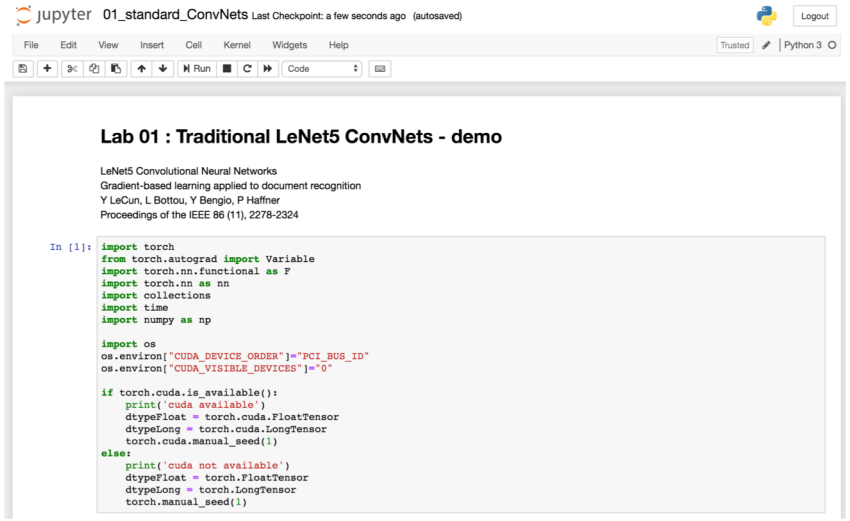
- Optimization

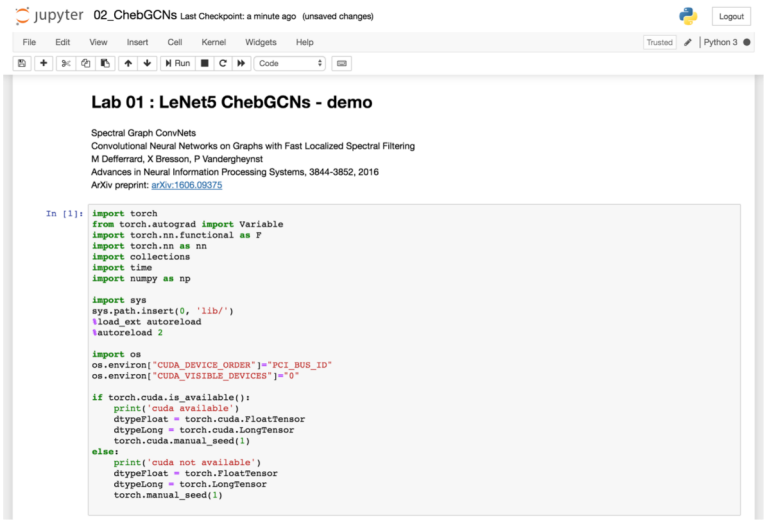




Lab 01

- Traditional LeNet5 ConvNets
- LeNet5 ChebGCNs





Lab 01

- ChebGCNs for your graph problems :

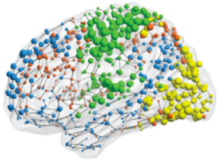
A is the adjacency matrix of your graph.

Graph Adjacency Matrix

```
from lib.grid_graph import grid_graph
from lib.coarsening import coarsen
from lib.coarsening import lmax_L
from lib.coarsening import perm_data
from lib.coarsening import rescale_L

# Construct graph
t_start = time.time()
grid_side = 28
number_edges = 8
metric = 'euclidean'


######## YOUR GRAPH ADJACENCY MATRIX HERE ########
A = grid_graph(grid_side,number_edges,metric) # create graph of Euclidean grid
######## YOUR GRAPH ADJACENCY MATRIX HERE ########


# Compute coarsened graphs
coarsening_levels = 4
L, perm = coarsen(A, coarsening_levels)
```

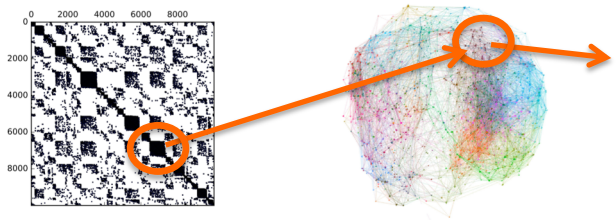
Outline

- Part 2: Spectral Graph ConvNets
  - Problem Setting
  - Graph Theory
    - Spectral Graph Convolution
    - Graph Pooling
  - Spectral GCNs
  - Applications

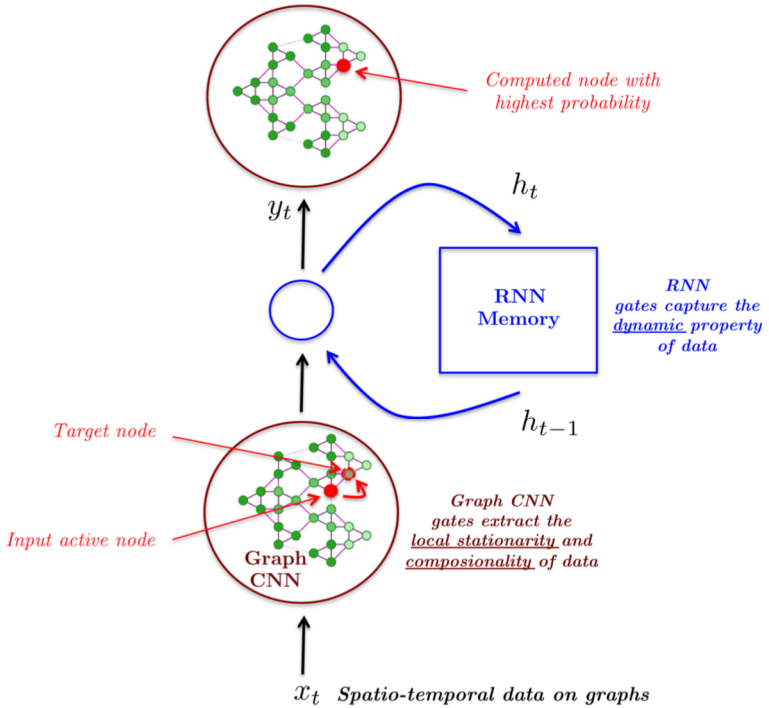


Temporal Predictions on Graphs[1]

- We combine ChebGCNs + LSTM :
  - ChebGCNs analyzes static information from data on graphs (find multi-scale local stationary patterns).
    - Words form a space of topics that are encoded with graphs of relationships.



  - LSTM identifies dynamical information (non-linear temporal properties of data).



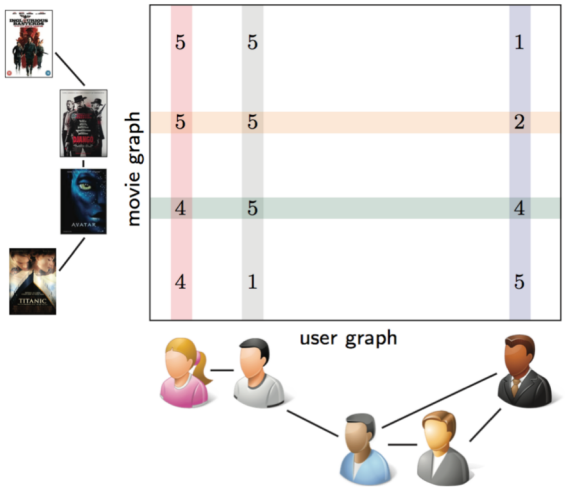
- We consider the basic NLP task of Word Prediction.

[1] Y Seo, M Defferrard, P Vandergheynst, X Bresson, Structured sequence modeling with graph convolutional recurrent networks, International Conference on Neural Information Processing, 2018

Recommender Systems

- Tackling the Netflix challenge.
  - Matrix Completion on Multiple Graphs[1] :

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times n}} \quad \|\mathbf{X}\|_* + \mu\|\boldsymbol{\Omega}\circ(\mathbf{X}-\mathbf{A})\|_{\mathrm{F}}^2 + \mu_{\mathrm{c}}\underbrace{\mathrm{tr}(\mathbf{X}\boldsymbol{\Delta}_{\mathrm{c}}\mathbf{X}^\top)}_{\|\mathbf{X}\|_{\mathcal{G}_{\mathrm{c}}}^2} + \mu_{\mathrm{r}}\underbrace{\mathrm{tr}(\mathbf{X}^\top\boldsymbol{\Delta}_{\mathrm{r}}\mathbf{X})}_{\|\mathbf{X}\|_{\mathcal{G}_{\mathrm{r}}}^2}$$



- Multi-graph convolutional layer[2] :

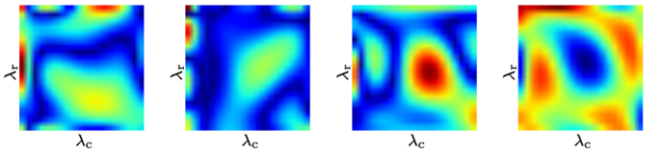
$$\mathbf{Y}_l^{(k)} = \xi\left(\sum_{l'=1}^{p}\sum_{j,j'=0}^{r}\theta_{jj'll'}T_j(\tilde{\boldsymbol{\Delta}}_{\mathrm{r}})\mathbf{Y}_{l'}^{(k-1)}T_{j'}(\tilde{\boldsymbol{\Delta}}_{\mathrm{c}})\right)$$

Chebyshev

2D-spectral filters

$l = 1,\dots,q$ #output features

$l' = 1,\dots,p$ #input features



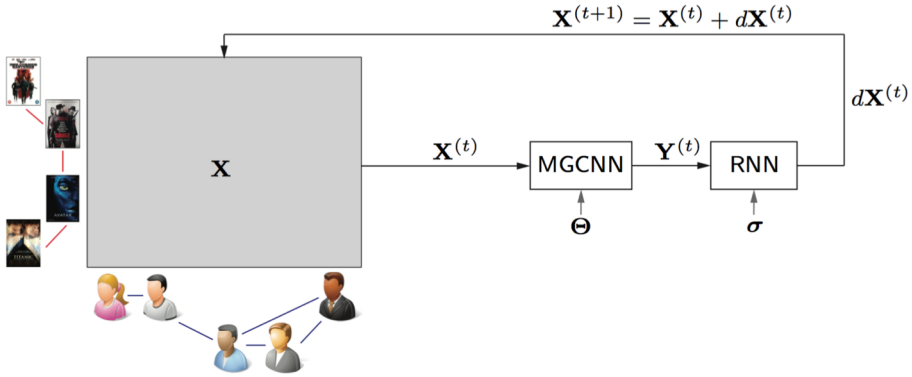
[1] V Kalofolias, X Bresson, M Bronstein, P Vandergheynst, Matrix completion on graphs, NeurIPS'14
[2] F Monti, M Bronstein, X Bresson, Geometric matrix completion with recurrent multi-graph neural networks, NeurIPS'17

Recommender Systems[1]

- We cast the completion problem as non-linear diffusion process.
  - Less parameters to learn (less overfitting).
  - Favorable for highly sparse entries (better diffusion).
  - Allows to learn temporal dynamics of entries if such information is available.

Method	MovieLens[1]	Flixster[2]	Douban[3]	Yahoo[4]
IMC[5]	1.653	–	–	–
GMC[6]	0.996	–	–	–
MC[7]	0.973	–	–	–
GRALS[8]	0.945	1.245	0.833	38.042
sRGCNN (Cheb)[9]	**0.929**	**0.926**	**0.801**	**22.415**
sRGCNN (Cayley)[10]	**0.922**	–	–	–

Data: [1]Miller et al. 2003; [2]Jamali, Ester 2010; [3]Ma et al. 2011; [4]Dror et al. 2012
Methods: [5]Jain, Dhillon 2013; [6]Kalofolias et al. 2014; [7]Candès, Recht 2012; [8]Rao et al. 2015; [9]Monti, Bresson, Bronstein 2017; [10]Levie et al. 2017



Recurrent multigraph CNN (RMCNN) architecture for matrix completion

$$\min_{\boldsymbol{\Theta},\boldsymbol{\sigma}} \|\mathbf{X}^{(T)}_{\boldsymbol{\Theta},\boldsymbol{\sigma}}\|^2_{\mathcal{G}_\mathrm{r}} + \|\mathbf{X}^{(T)}_{\boldsymbol{\Theta},\boldsymbol{\sigma}}\|^2_{\mathcal{G}_\mathrm{c}} + \frac{\mu}{2}\|\boldsymbol{\Omega} \circ (\mathbf{X}^{(T)}_{\boldsymbol{\Theta},\boldsymbol{\sigma}} - \mathbf{Y})\|^2_\mathrm{F}$$

[1] F Monti, M Bronstein, X Bresson, Geometric matrix completion with recurrent multi-graph neural networks, NeurIPS'17

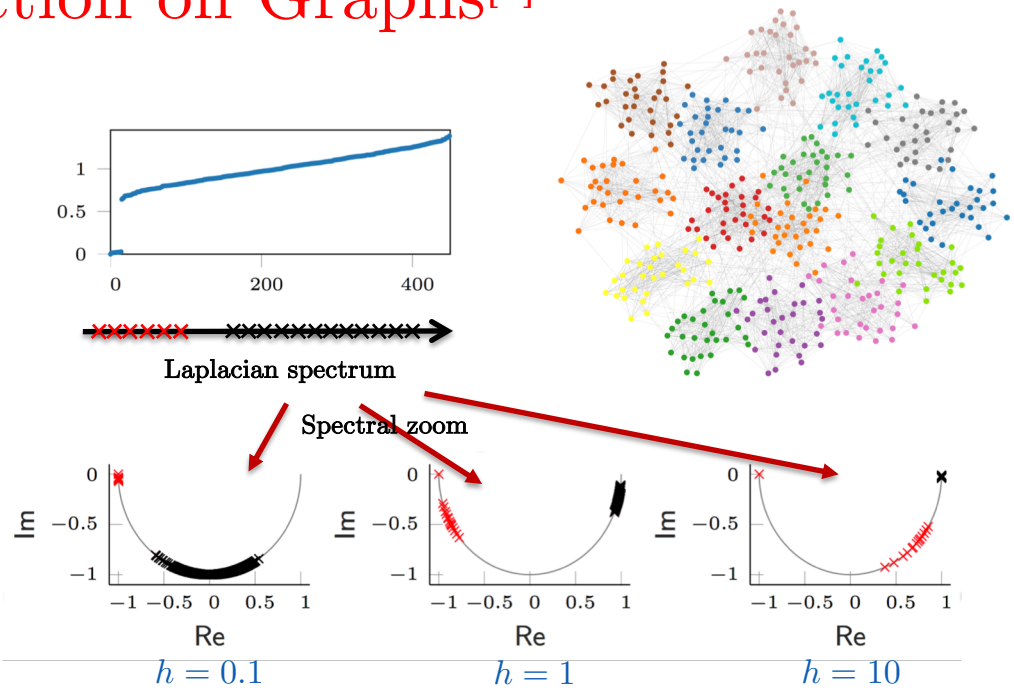
Community Detection on Graphs[1]

- ChebGCNs are unstable to produce filters with frequency bands of interest.
- We represent spectral filters with Cayley rationals of order r :

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

Coefficients (bypropagation)

Spectral zoom (hyper-parameter)



☺ CayleyGCNs w/ same properties of ChebGCNs in addition of

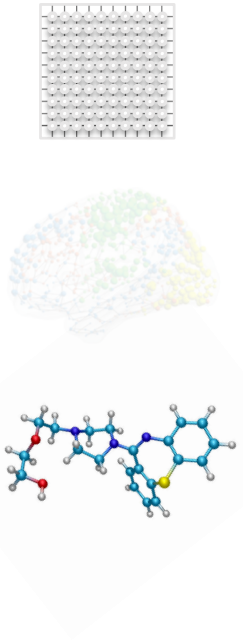
☺ Spectral zoom property allowing to better localize in frequency.

☺ Richer class of filters than Chebyshev for the same order.

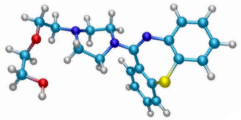
[1] R Levie, F Monti, X Bresson, MM Bronstein, Cayleynets: Graph convolutional neural networks with complex rational spectral filters, IEEE Transactions on Signal Processing, 2018

Outline

- Part 1: Traditional ConvNets
- Part 2: Spectral Graph ConvNets
- **Part 3: Spatial Graph ConvNets**
- Conclusion



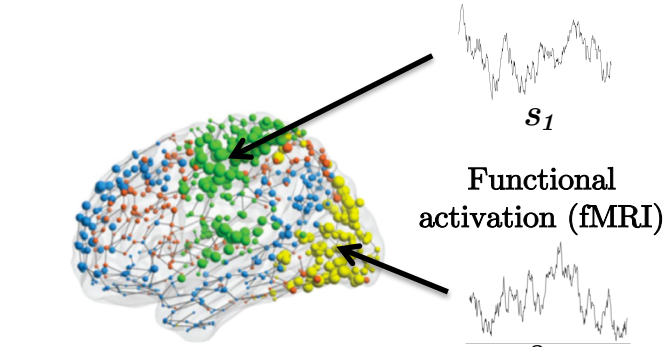
Outline



- Part 3: Spatial Graph ConvNets
  - Problem Setting
  - Spatial Graph Architectures
  - Quantum Chemistry
  - Operations Research
  - DGL

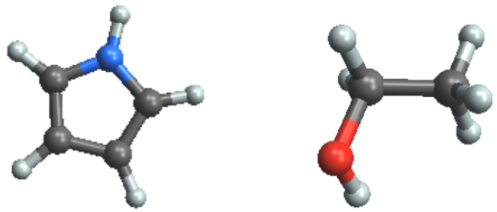
Problem Setting

- Spectral Graph ConvNets :
  - Given fixed graph(s) G, and a set of signals s_k on G to be analyzed.



Brain connectivity network (sMRI)
Fixed graph G

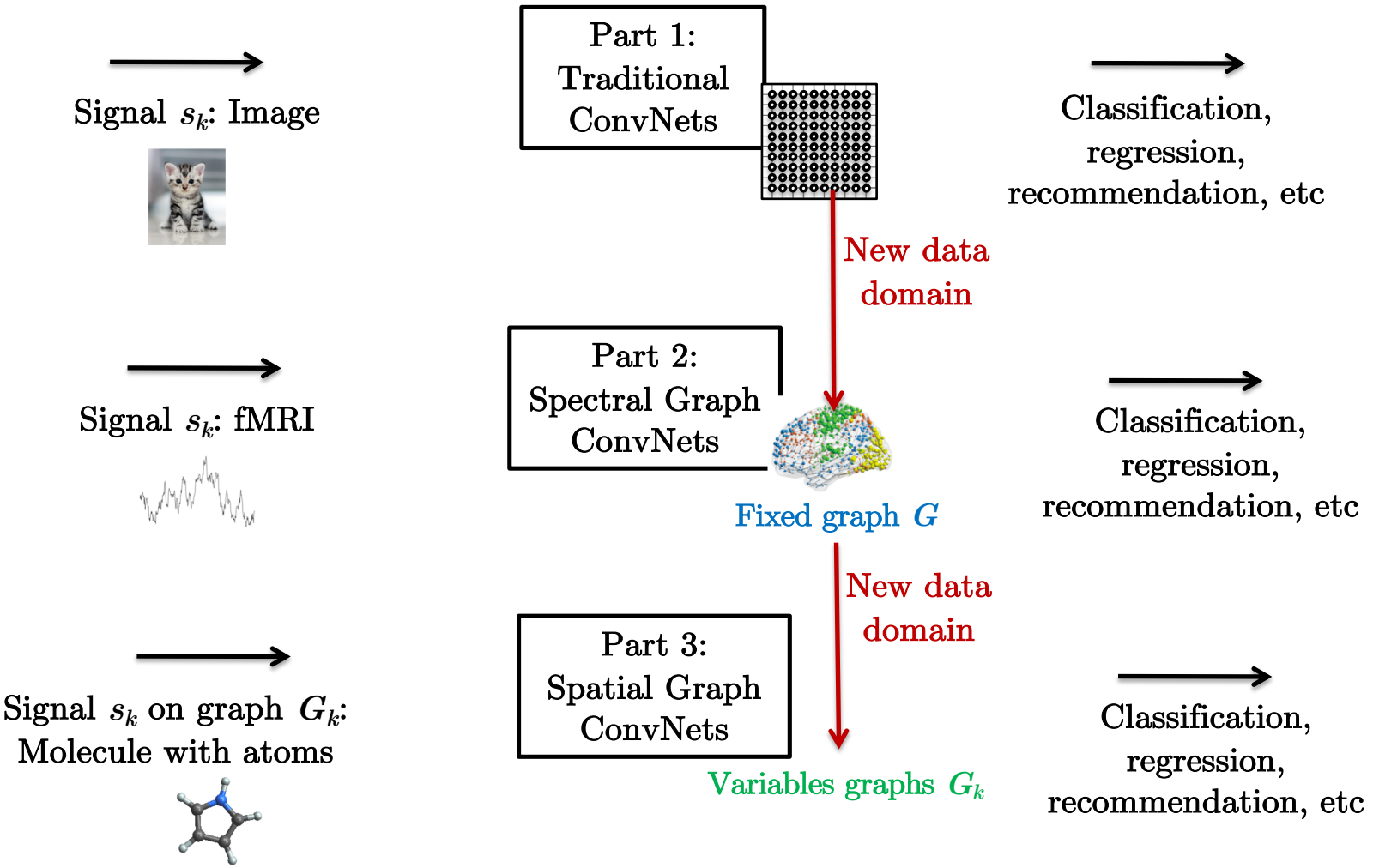
- Spatial Graph ConvNets :
  - Given a set of variable graphs G_k and signals s_k on G_k to be analyzed.



G_1, s_1 G_2, s_2

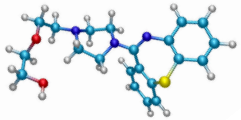
Molecules
Variable graphs G_k

Graph ConvNet Architectures



Outline

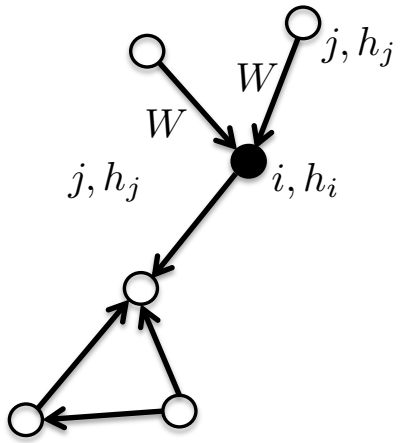
- Part 3: Spatial Graph ConvNets
  - Problem Setting
  - Spatial Graph Architectures
  - Quantum Chemistry
  - Operations Research
  - DGL



Spatial GNNs[1]

- Minimal inner structures to design spatial GNNs :
  - Invariant by vertex re-indexing (no graph matching is required)
  - Locality/local reception field (only neighbors are considered)
  - Weight sharing (convolutional operations)
  - Independence w.r.t. graph size

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

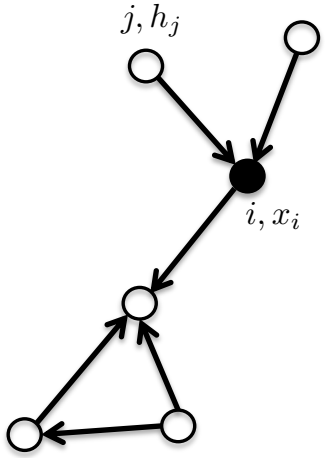


[1] Scarselli, Gori, Tsoi, Hagenbuchner, Monfardini, The Graph Neural Network Model, 2009
[2] Li, Tarlow, Brockschmidt, Zemel, Gated Graph Sequence Neural Networks, 2015

Graph RNNs

- Vanilla Graph RNNs with Multi-Layer Perceptron (MLP)[1] :

$$h_i = \sum_{j \to i} \mathcal{C}_{\text{G-MLP}}(x_i, h_j) = \sum_{j \to i} A\sigma(B\sigma(Ux_i + Vh_j))$$



- Graph GRUs[2,3] (Gated Recurrent Units) :

$$h_i = \mathcal{C}_{\text{G-GRU}}(x_i, \sum_{j \to i} h_j)$$

Fixed-point iterative scheme required
(unlike NLP, no directed acyclic graph) :

$$\begin{aligned}
\bar{h}_i^t &= \sum_{j \to i} h_j^t, \quad h_i^{t=0} = x_i \\
z_i^{t+1} &= \sigma(U_z h_i^t + V_z \bar{h}_i^t) \\
r_i^{t+1} &= \sigma(U_r h_i^t + V_r \bar{h}_i^t) \\
\tilde{h}_i^{t+1} &= \tanh\big(U_h(h_i^t \odot r_i^{t+1}) + V_h \bar{h}_i^t\big) \\
h_i^{t+1} &= (1 - z_i^{t+1}) \odot h_i^t + z_i^{t+1} \odot \tilde{h}_i^{t+1}
\end{aligned}$$

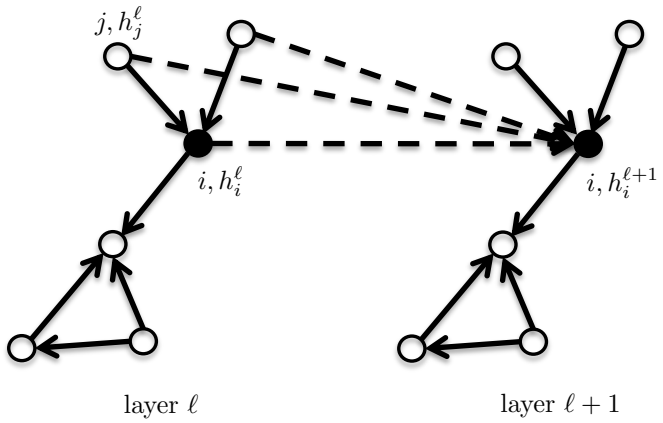
[1] Scarselli, Gori, Tsoi, Hagenbuchner, Monfardini, The Graph Neural Network Model, 2009
[2] Li, Tarlow, Brockschmidt, Zemel, Gated Graph Sequence Neural Networks, 2015
[3] Cho, Merrienboer, Gulcehre, Bahdanau, Bougares, Schwenk, Bengio, Learning Phrase Representations using RNN for Statistical Machine Translation, 2014

Vanilla Graph ConvNets

- Vanilla GCNs[1,2] (with ReLU) and GraphSAGE[3] (with max) :

$$h_i^{\ell+1} = \mathcal{C}_{\text{G-VCN}}\Big(h_i^\ell, \sum_{j \to i} h_j^\ell\Big), \quad h_i^{\ell=0} = x_i$$

$$= \text{ReLU}\Big(U^\ell h_i^\ell + V^\ell \sum_{j \to i} h_j^\ell\Big), \quad h_i^{\ell=0} = x_i$$



[1] Sukhbaatar, Szlam, Fergus, Learning Multiagent Communication with Backpropagation, 2016
[2] Kipf, Welling, Semi-Supervised Classification with Graph Convolutional Networks, 2017
[3] Hamilton, Ying, Leskovec, Inductive representation learning on large graphs, 2017

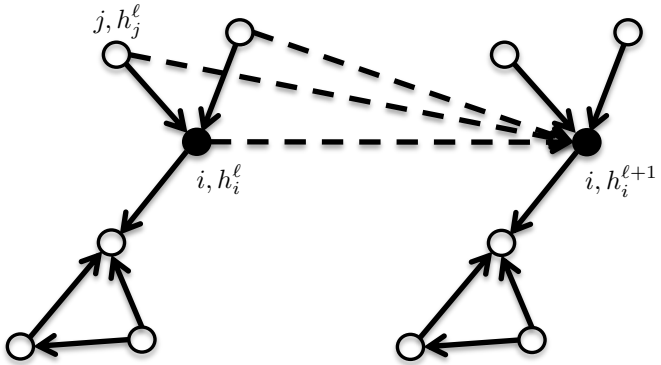
GatedGCNs[1]

- Graph ConvNets architecture[2,3,4] augmented with edge gating mechanism[5], residuality[6] and batch normalization[7] :

$$h_i^{\ell+1} = h_i^\ell + \mathrm{ReLU}\Big(\mathrm{BN}\Big(U^\ell h_i^\ell + \sum_{j\to i} \eta_{ij}^\ell \odot V^\ell h_j^\ell\Big)\Big)$$

edge gates
/anisotropic property
/dense attention

$$\eta_{ij} = \sigma\left(A^\ell h_i^\ell + B^\ell h_j^\ell\right)$$



Design the simplest learnable anisotropic and multiscale diffusion operator on graphs
[Perona-Malik'87 PDE inspiration]

[1] Bresson, Laurent, Residual gated graph convnets, 2017
[2] Sukhbaatar, Szlam, Fergus, Learning Multiagent Communication with Backpropagation, 2016
[3] Kipf, Welling, Semi-Supervised Classification with Graph Convolutional Networks, 2017
[4] Hamilton, Ying, Leskovec, Inductive representation learning on large graphs, 2017
[5] Marcheggiani, Titov, Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling, 2017
[6] He, Zhang, Ren, Sun, Deep Residual Learning for Image Recognition, 2016
[7] Ioffe, Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015

Graph Attention Networks[1]

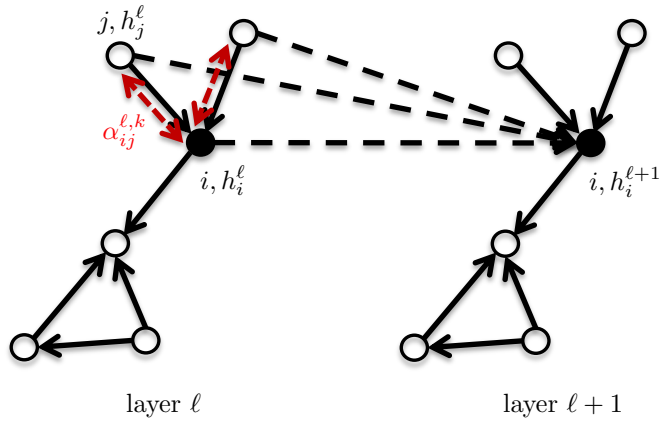
- Attention mechanism in 1-hop neighborhood :

$$h_i^{\ell+1} = \Big(\Big\|_{k=1}^{K} \sigma\Big(\sum_{j\to i} \alpha_{ij}^{\ell,k}\, V^{\ell,k} h_j^{\ell}\Big)\Big) W^{\ell}$$

$$\alpha_{ij}^{\ell,k} = \mathrm{softmax}_{\text{1-hop}}\big((Q^{\ell,k} h_i^{\ell})^T (K^{\ell,k} h_j^{\ell})\big)$$

$$= \frac{e^{(Q^{\ell,k} h_i^{\ell})^T (K^{\ell,k} h_j^{\ell})}}{\sum_{j'\to i} e^{(Q^{\ell,k} h_i^{\ell})^T (K^{\ell,k} h_{j'}^{\ell})}}$$

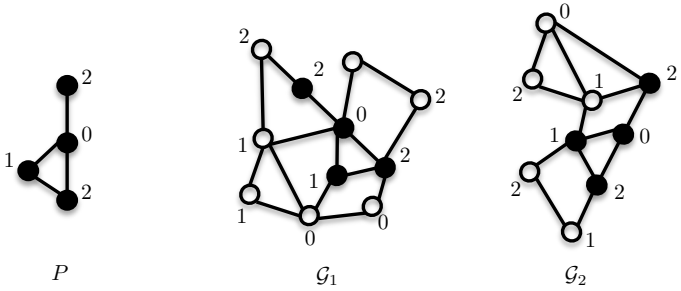
Self-attention mechanism
in 1-hop neighborhood
/sparse attention

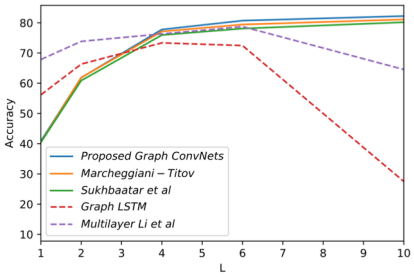


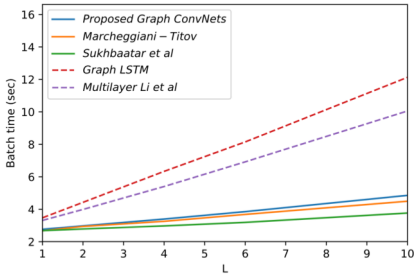
[1] Velickovic, Cucurull, Casanova, Romero, Lio, Bengio, Graph Attention Networks, 2018

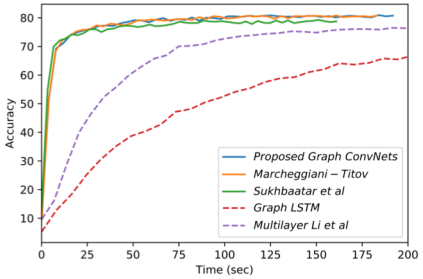
GraphRNNs vs GCNs[1]

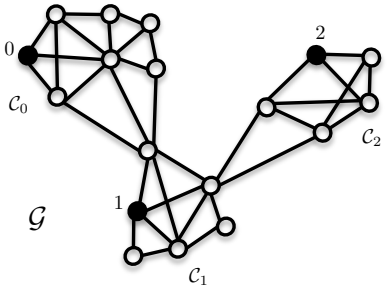
- Numerical comparison on two basic graph problems:
  - Sub-graph matching/recognition[2]
  - Semi-supervised classification











- GCNs offer better performances than GraphRNNs (as they can stack many layers).

[1] Bresson, Laurent, Residual gated graph convnets, 2017
[2] Scarselli, Gori, Tsoi, Hagenbuchner, Monfardini, The Graph Neural Network Model, 2009

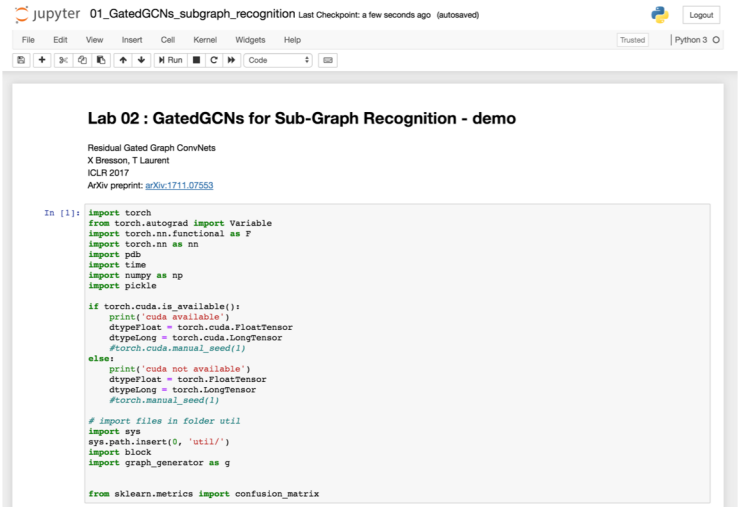
Anisotropy vs Isotropy

- Standard ConvNets produce anisotropic filters because Euclidean grids have directional structure.
- Graph ConvNets compute isotropic filters because there is no notion of directions on arbitrary graphs.
- How to get anisotropy back in GNNs ?
  - Edge gates[1]/attention mechanism[2] information to treat neighbors differently.
  - Differentiate graph edges[3] (e.g. different connections between atoms)

[1] Bresson, Laurent, Residual gated graph convnets, 2017
[2] Velickovic, Cucurull, Casanova, Romero, Lio, Bengio, Graph Attention Networks, 2018
[3] Gilmer, Schoenholz, Riley, Vinyals, Dahl, Neural message passing for quantum chemistry, 2017

Lab 02

- GatedGCNs for Sub-Graph Recognition
- GatedGCNs for Graph Classification


jupyter 01_GatedGCNs_subgraph_recognition Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

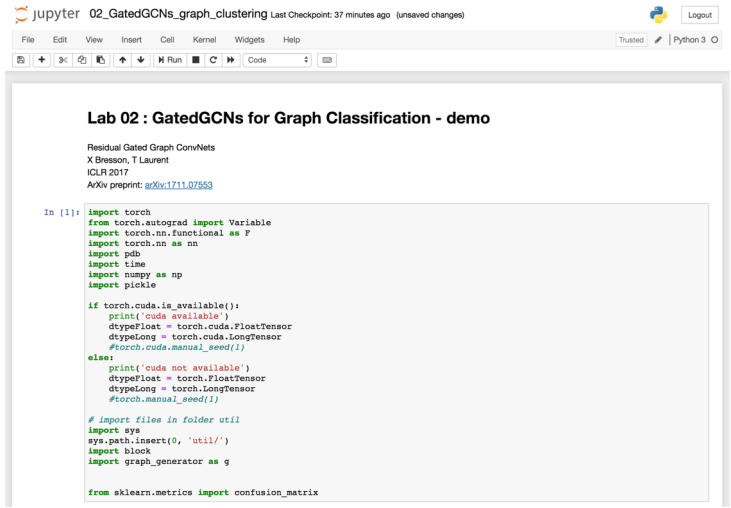
Run Code

Lab 02 : GatedGCNs for Sub-Graph Recognition - demo

Residual Gated Graph ConvNets
X Bresson, T Laurent
ICLR 2017
ArXiv preprint: arXiv:1711.07553

In [1]:

```
import torch
from torch.autograd import Variable
import torch.nn.functional as F
import torch.nn as nn
import pdb
import time
import numpy as np
import pickle

if torch.cuda.is_available():
    print('cuda available')
    dtypeFloat = torch.cuda.FloatTensor
    dtypeLong = torch.cuda.LongTensor
    #torch.cuda.manual_seed(1)
else:
    print('cuda not available')
    dtypeFloat = torch.FloatTensor
    dtypeLong = torch.LongTensor
    #torch.manual_seed(1)

# import files in folder util
import sys
sys.path.insert(0, 'util/')
import block
import graph_generator as g


from sklearn.metrics import confusion_matrix
```


jupyter 02_GatedGCNs_graph_clustering Last Checkpoint: 37 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

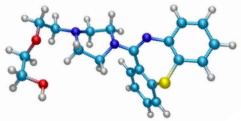
Lab 02 : GatedGCNs for Graph Classification - demo

Residual Gated Graph ConvNets
X Bresson, T Laurent
ICLR 2017
ArXiv preprint: arXiv:1711.07553

In [1]:

```
import torch
from torch.autograd import Variable
import torch.nn.functional as F
import torch.nn as nn
import pdb
import time
import numpy as np
import pickle

if torch.cuda.is_available():
    print('cuda available')
    dtypeFloat = torch.cuda.FloatTensor
    dtypeLong = torch.cuda.LongTensor
    #torch.cuda.manual_seed(1)
else:
    print('cuda not available')
    dtypeFloat = torch.FloatTensor
    dtypeLong = torch.LongTensor
    #torch.manual_seed(1)

# import files in folder util
import sys
sys.path.insert(0, 'util/')
import block
import graph_generator as g


from sklearn.metrics import confusion_matrix
```

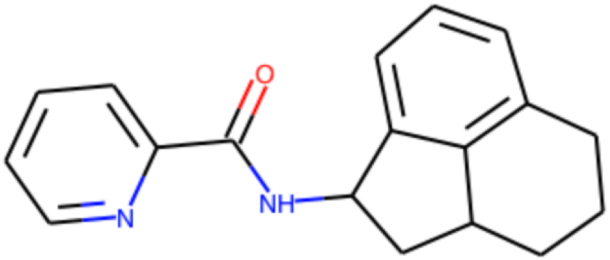
Outline



- Part 3: Spatial Graph ConvNets
  - Problem Setting
  - Spatial Graph Architectures
  - Quantum Chemistry
  - Operations Research
  - DGL

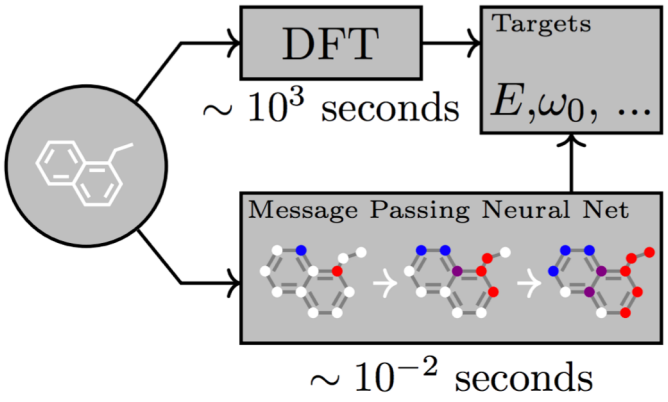
Quantum Chemistry

- Molecules are graphs where vertices are atoms (e.g. H, C, N, Cl) and edges are types of bonds between atoms (e.g. single, double, aromatic).
- Tasks are :
  - Computing chemical properties :
    - Solubility
    - Octanol-water partition coefficient
    - Synthetic accessibility
  - Generating new molecules with specific optimized properties



Molecule Regression

- Duvenaud-et-al NeurIPS'15 :
  - Learn molecule fingerprints
- Gilmer-Vinyals-Dahl-et.al 2017 (Google) :
  - Use graph NNs to learn 13 important chemical properties.
  - Standard estimation is done by DFT (approximation of coupled Schrodinger PDEs).

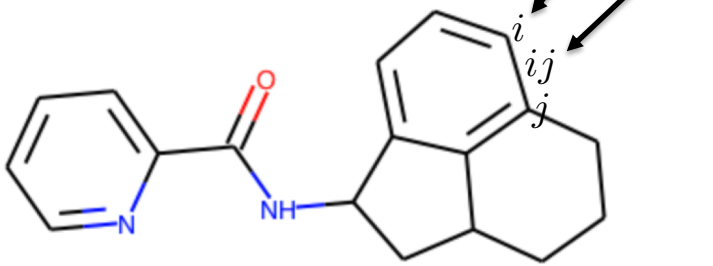


Chemical Properties	enn-s2s Gilmer-etal'16 Google	SchNet Schutt-etal'17 TU Berlin	Message Passing Jørgensen-etal'18 U. Danemark
alpha (Bohr^3)	0.092	0.235	0.077
Cv (cal/molK)	0.040	0.033	0.032
G (meV)	19	14	12.200
Gap (meV)	69	63	58.000
H (meV)	17	14	11.300
HOMO (meV)	43	41	36.700
LUMO (meV)	37	34	30.800
R^2 (Bohr^2)	0.180	0.073	0.072
U (meV)	19	19	10.600
U0 (meV)	19	14	10.500
Zpve (meV)	1.5	1.7	1.490
mu (Debye)	0.030	0.033	0.029

Table 7: Mean absolute error of predictions for different target properties of the QM9 dataset using 110k molecules.

Molecule Property Regression with GatedGCNs

- Two steps :
  - Vertex, edge and graph embedding layers
  - Regression layer



$$h_i^{\ell+1} = h_i^\ell + \mathrm{ReLU}\Big(\mathrm{BN}\Big(U^\ell h_i^\ell + \sum_{j\to i} \eta_{ij}^\ell \odot V^\ell h_j^\ell\Big), \quad h_i^{\ell=0} = \text{ atom\_type}$$

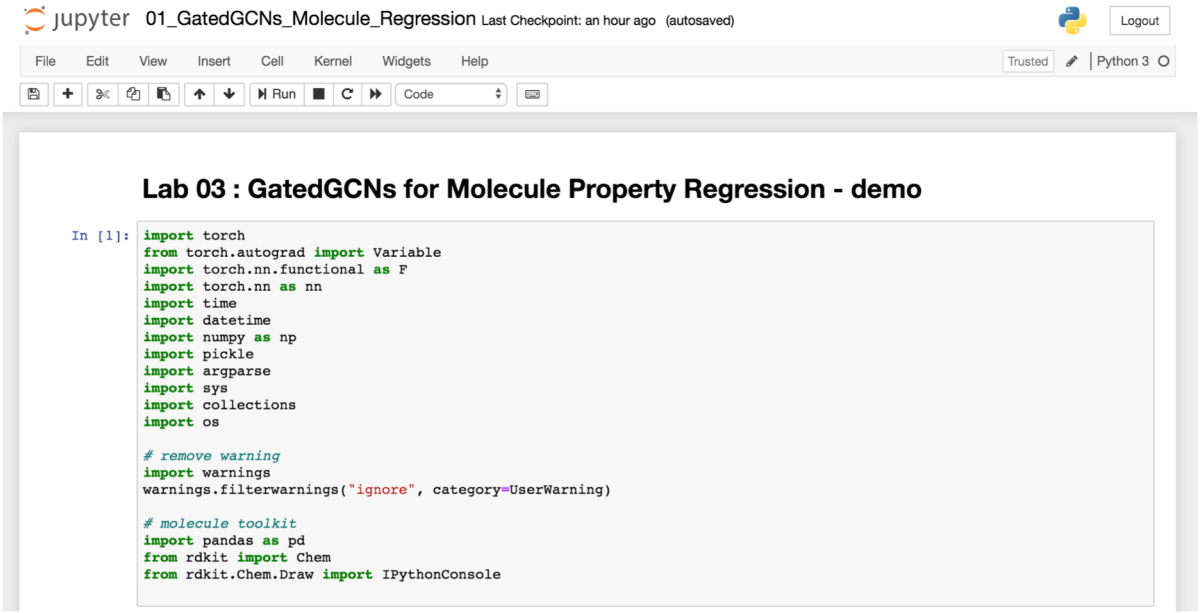
$$e_{ij}^{\ell+1} = e_{ij}^\ell + \mathrm{ReLU}\Big(\mathrm{BN}\Big(A^\ell h_i^{\ell+1} + B^\ell h_j^{\ell+1} + C^\ell e_{ij}^\ell\Big), \quad e_{ij}^{\ell=0} = \text{ bond\_type}$$

$$z = \sum_{ij} \gamma_{ij} \odot W e_{ij}^L \qquad \text{Molecule embedding}$$

$$p = \mathrm{MLP}(z) \qquad \text{Molecule property}$$

Lab 03

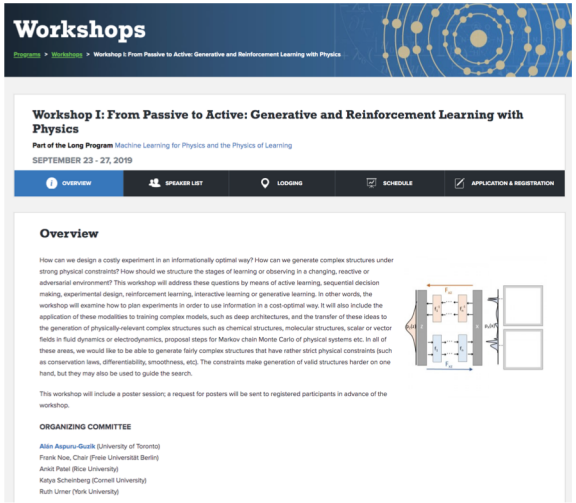
- GatedGCNs for Molecule Property Regression

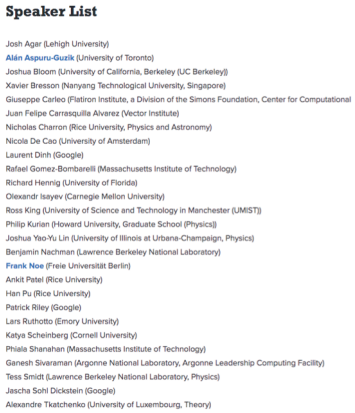


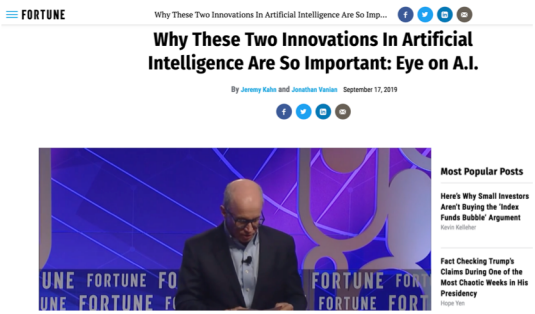
AI For Materials and Drugs Design

- Workshop "Generative and Reinforcement Learning with Physics" at IPAM-UCLA, Sept 2019
  - Apply Deep Learning to generate new materials and drugs (big impact in the coming years)
  - Self-driving lab to create these materials and drugs quicker and less expensively (takes a decade and costs $350M-$2.7B to bring a new drug to market)

    https://fortune.com/2019/09/17/artificial-intelligence-insilico-deepmind

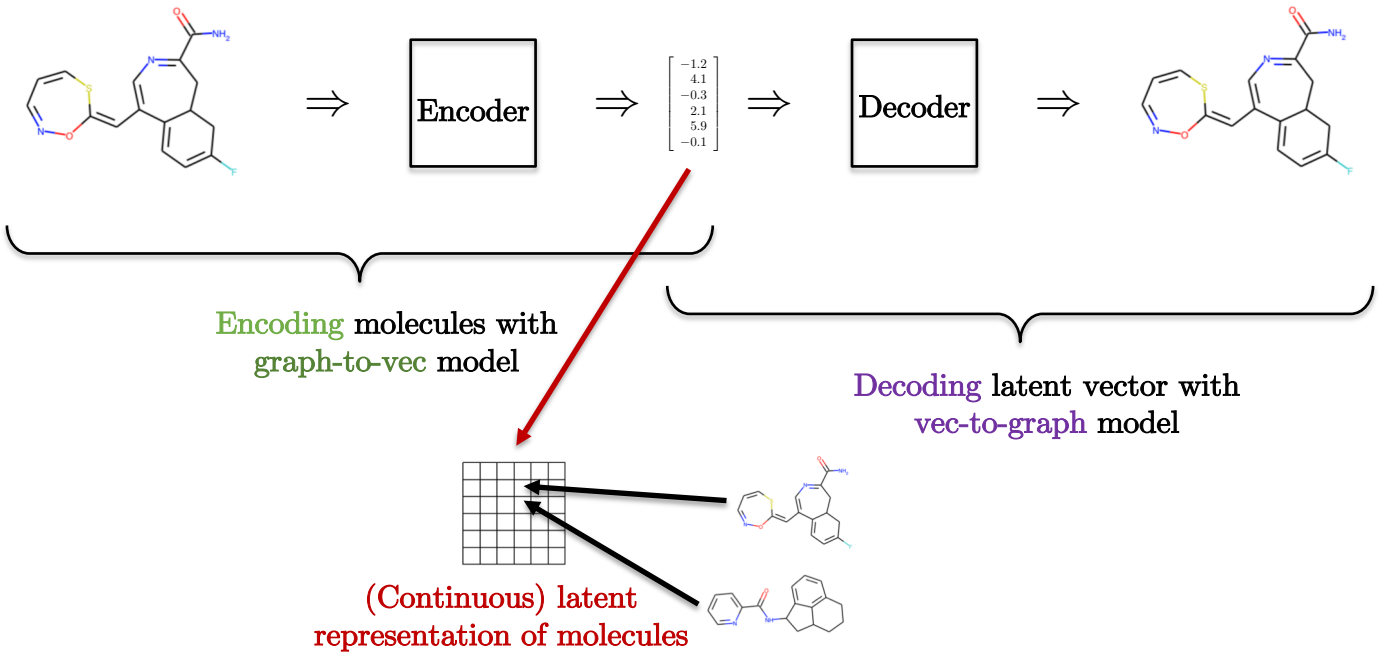






Graph Molecular Generation

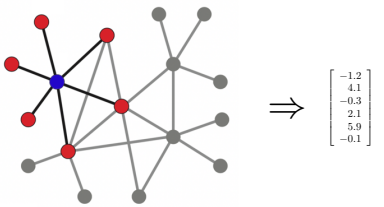
- Goal is to design a neural network that can auto-encode molecules, generate novel molecules, produce molecules with optimized chemical property.
- Graph VAE Architecture with GatedGCNs[1] :



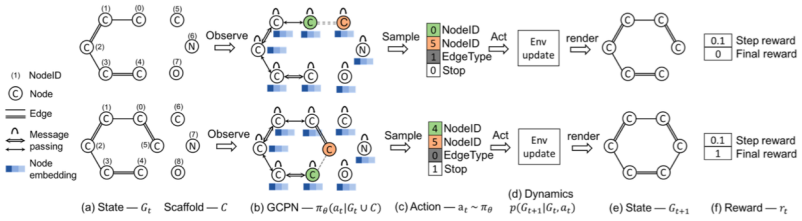
[1] Bresson, Laurent, A Two-Step Graph Convolutional Decoder for Molecule Generation, Workshop ML and Physics at NeurIPS'19

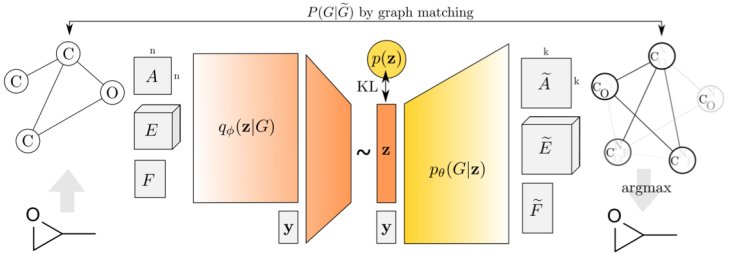
Graph Encoder and Decoder

- Graph NNs[1,2] used to encode molecules to predict molecular properties (1-2 orders of magnitude faster than DFT).
  - Encoding is easy. Decoding is more challenging !
- Two decoding approaches :
  - Auto-regressive models : Sequential generation of molecules (atom-by-atom)[3,4].
  - One-shot models : Generation of all atoms and bonds in a single pass[5,6].



$$h_i = f_{\text{node}}(\{h_j\}, j \in \mathcal{N}(i))$$
$$z = g_{\text{graph}}(\{h_i\}, i \in V)$$





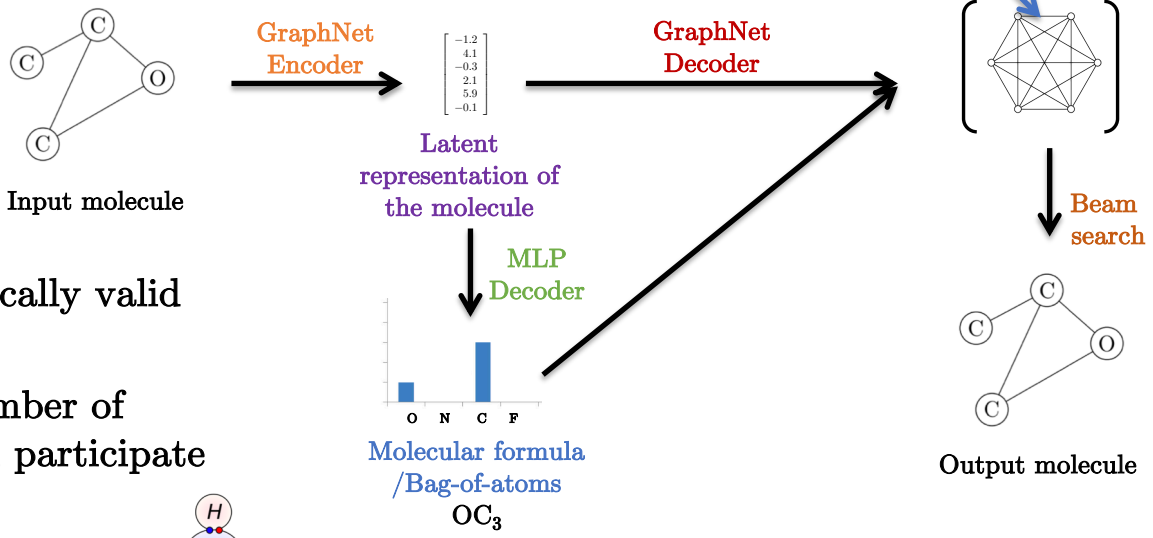
[1] Duvenaud, Maclaurin, Iparraguirre, Bombarell, Hirzel, Aspuru-Guzik, Adams, Convolutional networks on graphs for learning molecular fingerprints, 2015
[2] Gilmer, Schoenholz, Riley, Vinyals, Dahl, Neural message passing for quantum chemistry, 2017
[3] Jin, Barzilay, Jaakkola, Junction Tree Variational Autoencoder for Molecular Graph Generation, 2018
[4] You, Liu, Ying, Pande, Leskovec, Graph convolutional policy network for goal-directed molecular graph generation, 2018
[5] Simonovsky, Komodakis, GraphVAE: Towards generation of small graphs using variational autoencoders, 2018
[6] De Cao, Kipf, MolGAN: An implicit generative model for small molecular graphs, 2018

Single-Shot Molecular VAE

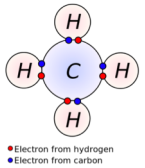
- It is hard to generate simultaneously atoms and bonds between atoms.
  - We propose to disentangle these problems with :
    - Atom generation with molecular formula.
    - Bond generation using molecular formula and decoder.
  - Advantages:
    - Simple and fast

      (GPU parallelizable) vs

      auto-regressive models.



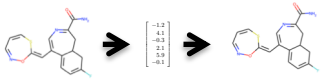
- The one-shot model may not produce a chemically valid molecule:
  - Violation of atom valency (maximum number of electrons in the outer atom shell that can participate of a chemical bond).
  - We use beam search to produce valid molecules (borrowed from NLP sequence generation).



Proposed molecular (variational) auto-encoder system

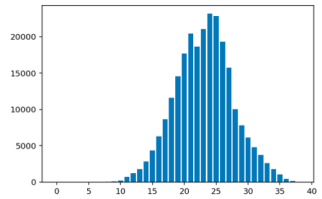
Numerical Experiments

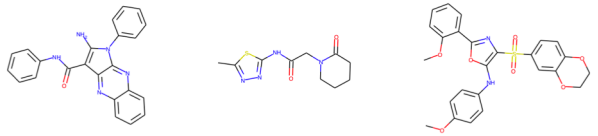
- Molecule reconstruction
  - How many molecules are correctly decoded?



Method	Reconstruction	Validity
CVAE [Gomez-Bombarelli et al., 2016]	44.6%	0.7%
GVAE [Kusner et al., 2017]	53.7%	7.2%
SD-VAE [Dai et al, 2018]	76.2%	43.5%
GraphVAE [Simonovsky, Komodakis, 2018]	-	13.5%
JT-VAE (SL) [Jin et al, 2018]	76.7%	**100.0%**
GCPN (GAN+RL) [You et al, 2018]	-	-
OURS (VAE+SL)	**90.5%**	**100.0%**

Table 1: Percentage of successful reconstruction of 250k ZINC molecules.





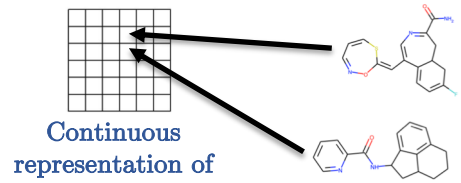
ZINC (250k drug like molecules, up to 38 heavy atoms)

- Molecule novelty
  - Beyond memorization – how many molecules sampled from the learned distribution are not in the training set?

Method	Novelty	Uniqueness
JT-VAE (SL) [Jin et al, 2018]	**100.0%**	**100.0%**
GCPN (GAN+RL) [You et al, 2018]	-	-
OURS (VAE+SL)	**100.0%**	**100.0%**

Table 2: Sample 5000 molecules from learned prior distribution.

Numerical Experiments



- Molecule optimization
  - How much property improvement can we obtain when optimizing in the latent space?
  - Optimization of solubility constrained by the similarity between the original molecule and the new generated molecule.

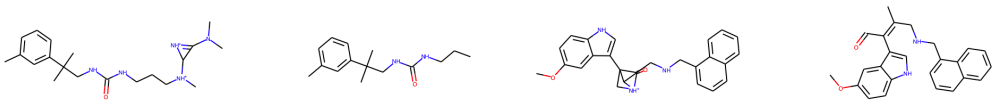
	JT-VAE [Jin et al, 2018] (SL)			GCPN [You et al, 2018] (GAN+RL)			OURS (VAE+SL)		
δ	Improvement	Similarity	Success	Improvement	Similarity	Success	Improvement	Similarity	Success
0.0	1.91 ± 2.04	0.28 ± 0.15	97.5%	4.20 ± 1.28	**0.32 ± 0.12**	**100.0%**	**5.24 ± 1.55**	0.18 ± 0.12	**100.0%**
0.2	1.68 ± 1.85	0.33 ± 0.13	97.1%	4.12 ± 1.19	**0.34 ± 0.11**	**100.0%**	**4.29 ± 1.57**	0.31 ± 0.12	98.6%
0.4	0.84 ± 1.45	**0.51 ± 0.10**	83.6%	2.49 ± 1.30	0.47 ± 0.08	**100.0%**	**3.05 ± 1.46**	**0.51 ± 0.10**	84.0%
0.6	0.21 ± 0.71	**0.69 ± 0.06**	46.4%	0.79 ± 0.63	0.68 ± 0.08	**100.0%**	**2.46 ± 1.27**	0.67 ± 0.05	40.1%

Table 7: Molecule optimization results.

Molecule similarity 0.4

604: -2.94 | 604: 3.39 | 103: -5.40 | 103: 0.88

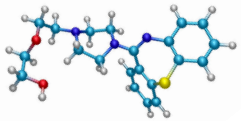
Molecule similarity 0.6



89: -5.64 | 89: 0.94 | 782: -2.57 | 782: 2.44

Outline

- Part 3: Spatial Graph ConvNets
  - Problem Setting
  - Spatial Graph Architectures
  - Quantum Chemistry
  - Operations Research
  - DGL

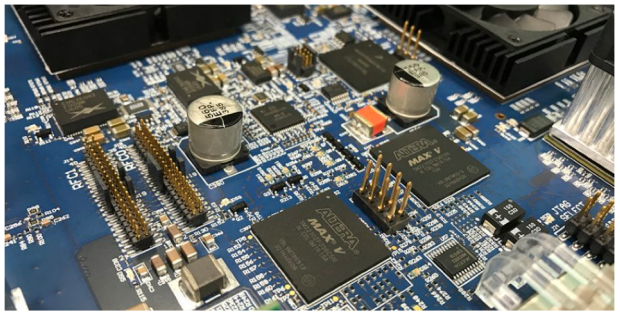


Operations Research

- OR/combinatorial problems s.a. assignment, routing, planning, supply chain, scheduling are used every day in revenue management, transportation, manufacturing, supply chain, public policy, hardware design, etc.

Amazon warehouse management

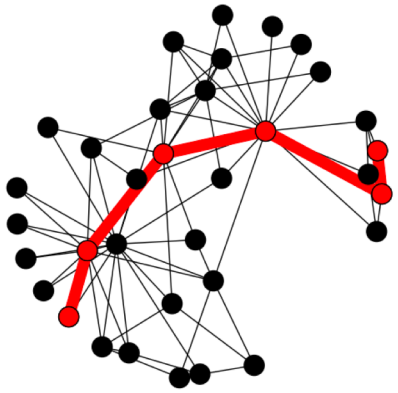
Uber taxis allocation



Hardware pieces placement

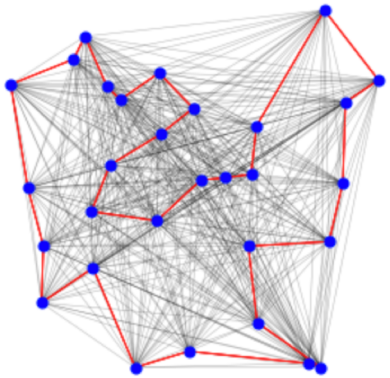
Easy Combinatorial Problems

- Shortest Path Problem :
  - Find a path between two vertices in a graph with shortest distance.
  - Dijkstra's algorithm : Exact solver in polynomial time O(E+VlogV) with Fibonacci heap.
  - Applications : Road Navigator, P2P network, etc



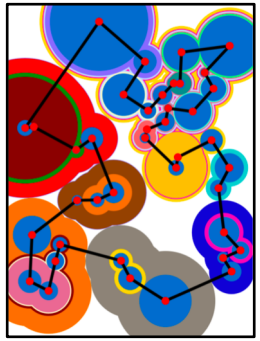
Hard Combinatorial Problems

- Travelling Salesman Problem (TSP) :
  - Find the shortest tour that visits each node and returns to the starting node.
  - NP-hard problem in combinatorial optimization.
    - Exact solution in factorial time O($V!$)
  - Long history : Von Neumann'51, Dantzig'54, Bellman-Held-Karp'62, Edmonds'67, etc.
    - William Cook (Waterloo U.), MAA Invited Talk on TSP, Jan. 2018 : https://www.youtube.com/watch?v=q8nQTNvCrjE
  - Applications : Transportation, scheduling, hardware design, etc.



TSP

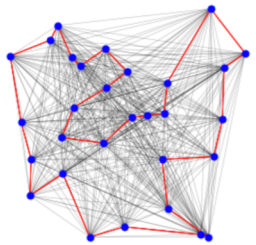
- State-of-the-art solution
  - Concorde[1] solver developed by Applegate, Bixby, Chvatal, Cook, 2006 (test-of-time).
  - Code available :
    - http://www.math.uwaterloo.ca/tsp/concorde
  - Leveraged 80 years of theoretical developments, data structures and heuristics from computer science.
  - Branch-and-Cut approach :
    - Cutting plane algorithms (Dantzig et al., 1954) to iteratively solve linear relaxations of TSP.
    - Branch-and-bound approach to reduce solution search space.



[1] Applegate, Bixby, Chvatal, Cook, 2006

Operations Research and Deep Learning

- Most OR problems are difficult/NP-hard, require heuristic solutions with years of research work and significant specialized knowledge.
  - Popular TSP problem has been studied for more than 80 years w/ von Neumann, Dantzig, etc. Best TSP solver is Concorde[1] (solve linear relaxations of TSP w/ branch-and-bound).
  - Any small change like adding constraints in TSP would require a new development of the specialized algorithm. OR designs hand-crafted algorithms.
- DL may lead to a potential breakthrough in traditional OR !
  - DL can learn universal high-quality algorithms to solve any OR problem by adapting to any continuous/discrete structure and by continuously learning by reinforcement and optimization of the cost under any constraint[2,3].



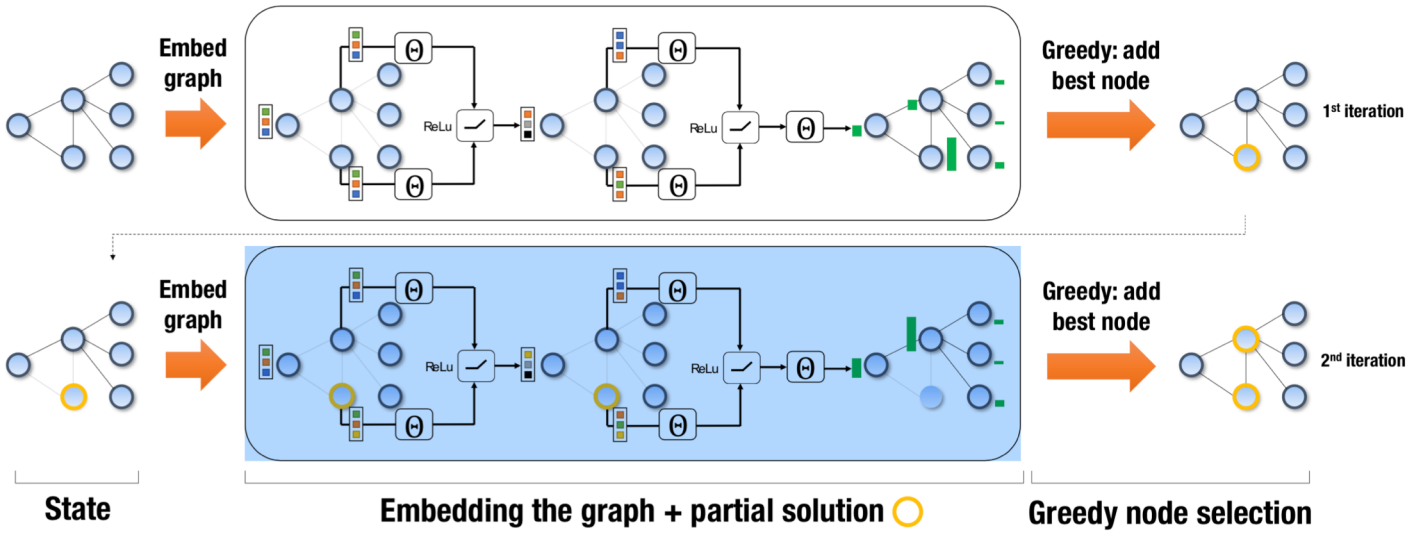
[1] Applegate, Bixby, Chvatal, Cook, 2006
[2] Bengio, Lodi, Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, 2018
[3] Bello, Pham, V Le, Norouzi, Bengio, Neural combinatorial optimization with reinforcement learning, 2016

Neural Networks for TSP

- Recent efforts to design learning algorithms for TSP (proof-of-concept).
- Two approaches :
  - Auto-regressive models : Sequential generation of TSP solution.
  - One-shot models : Generation of edge probability to be in TSP solution + sampling.

Auto-regressive NNs

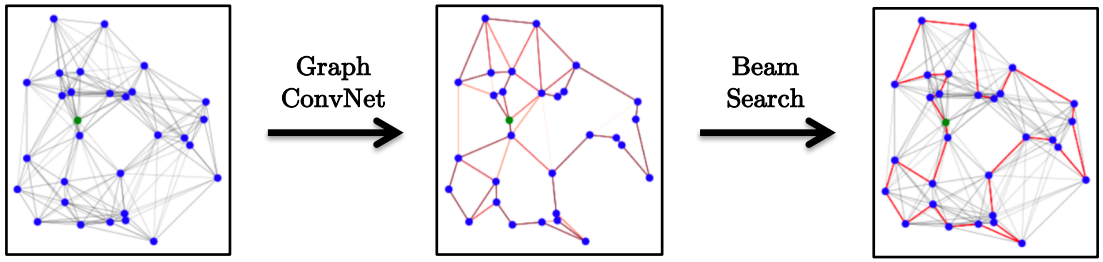
- Auto-regressive models : Sequential generation of TSP solution[1,2,3,4].



[1] Vinyals, Fortunato, Jaitly, Pointer networks, 2015
[2] Bello, Pham, V Le, Norouzi, Bengio, Neural combinatorial optimization with reinforcement learning, 2016
[3] Dai, Khalil, Zhang, Dilkina, Song, Learning combinatorial optimization algorithms over graphs, 2017
[4] Kool, van Hoof, Welling, Attention, learn to solve routing problems!, 2019

One-shot NNs

- One-shot models : Generation of edge probability to be in TSP solution + sampling[1].
  - GatedGCNs to predict edge probability + beam search to sample valid solution[2]



[1] Nowak, Villar, Bandeira, Bruna, A note on learning algorithms for quadratic assignment with graph neural networks, 2017
[2] CK Joshi, T Laurent, X Bresson, An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem, Workshop on GNNs at NeurIPS'19

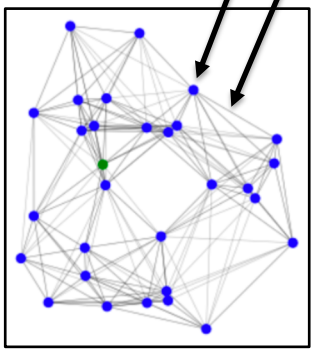
GatedGCNs for TSP[1,2]

Node and edge representations

$$h_i^{\ell+1} = h_i^\ell + \mathrm{ReLU}\Big(\mathrm{BN}\Big(W_1^\ell h_i^\ell + \sum_{j\sim i} \eta_{ij}^\ell \odot W_2^\ell h_j^\ell\Big)\Big) \quad \text{with} \quad \eta_{ij}^\ell = \frac{\sigma(e_{ij}^\ell)}{\sum_{j'\sim i}\sigma(e_{ij'}^\ell)+\varepsilon}$$

Dense attention

$$e_{ij}^{\ell+1} = e_{ij}^\ell + \mathrm{ReLU}\Big(\mathrm{BN}\Big(V_1^\ell e_{ij}^\ell + V_2^\ell h_i^\ell + V_3^\ell h_j^\ell\Big)\Big)$$



$$h_i^{\ell=0} = x_i \in \mathbb{R}^2$$

$$e_{ij}^{\ell=0} = \|x_i - x_j\|_2$$

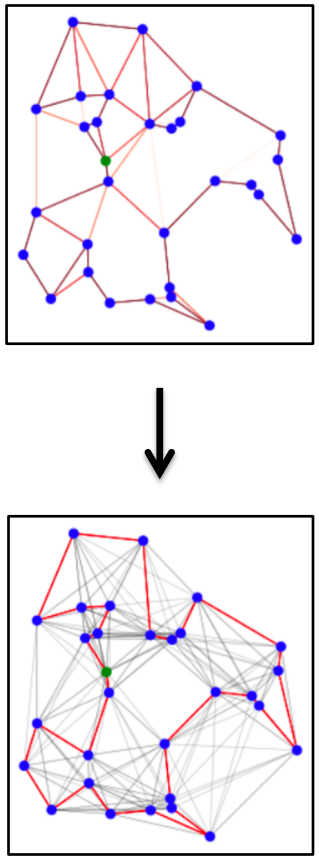
Edge classifier

$$p_{ij} = \mathrm{sigmoid}(\mathrm{MLP}(e_{ij}^L))$$ with loss being the logistic regression on edges.

[1] Bresson, Laurent, Residual gated graph convnets, ICLR'17
[2] CK Joshi, T Laurent, X Bresson, An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem, Workshop on GNNs at NeurIPS'19

Beam Search Decoder

- Output of GNN :
  - Edge probability function over the adjacency matrix of tour connections.
- Beam search decoder (NLP) :
  - Starting from a random node, expand b most probable edge connections among the node's neighbors.
  - Keep expanding the top-b partial tours at each stage until all nodes are visited.
  - Final prediction is the tour with the smallest length among the b tours at the end of search.



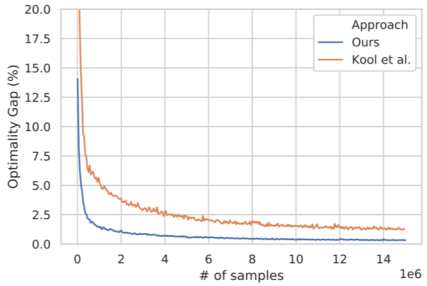
Numerical Experiments

- Our system :
  - Outperforms previous NNs in terms of both optimality gap and evaluation time (graph ConvNet and beam search are parallelizable).
  - But does not outperform Concorde.

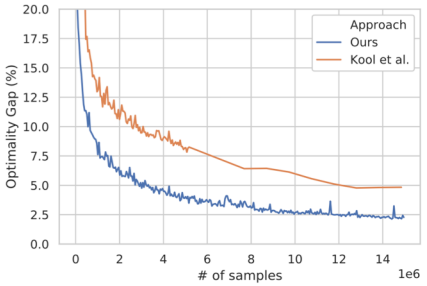
$$\text{gap} = \frac{1}{m}\sum_{i=1}^{m}\left(l_m/\hat{l}_m - 1\right)$$

	Method	Type	TSP20 Tour Len.	TSP20 Opt. Gap.	TSP20 Time	TSP50 Tour Len.	TSP50 Opt. Gap.	TSP50 Time	TSP100 Tour Len.	TSP100 Opt. Gap.	TSP100 Time
OR techniques	Concorde	Solver	3.84	0.00%	(1m)	5.70	0.00%	(2m)	7.76	0.00%	(3m)
	Gurobi	Solver	3.84	0.00%	(7s)	5.70	0.00%	(2m)	7.76	0.00%	(17m)
	OR Tools	H, S	3.85	0.37%		5.80	1.83%		7.99	2.90%	
NN techniques	GNN [Nowak et al., 2017]	SL, BS	3.93	2.46%			-			-	
	PtrNet [Bello et al., 2016]	RL, S		-		5.75	0.95%		8.00	3.03%	
	GAT [Kool et al., 2019]	RL, S	3.84	0.08%	(5m)	5.73	0.52%	(24m)	7.94	2.26%	(1h)
	GCN (Ours)	**SL, BS***	**3.84**	**0.01%**	**(12m)**	**5.70**	**0.01%**	**(18m)**	**7.87**	**1.39%**	**(40m)**

Sample efficiency



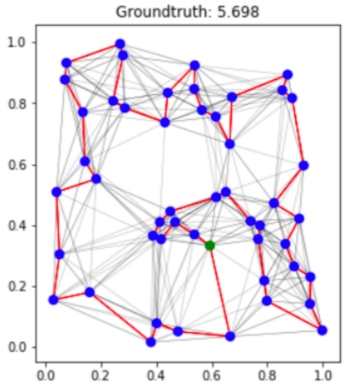
TSP50

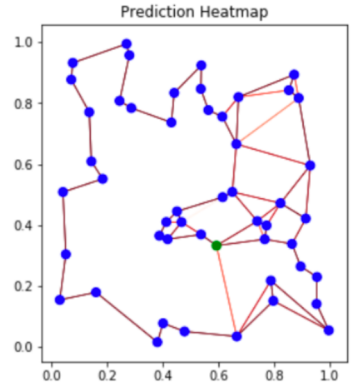


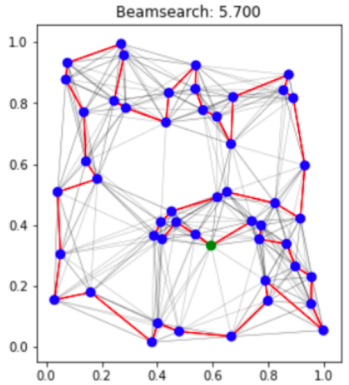
TSP100

Visualization

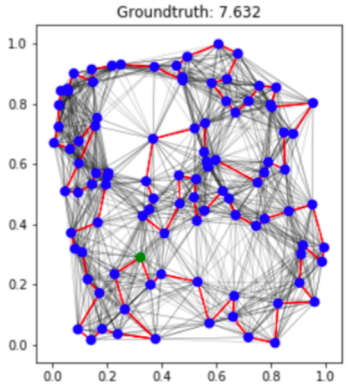
TSP50

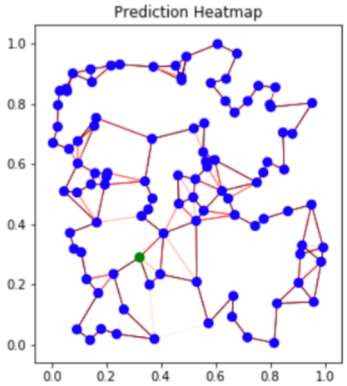


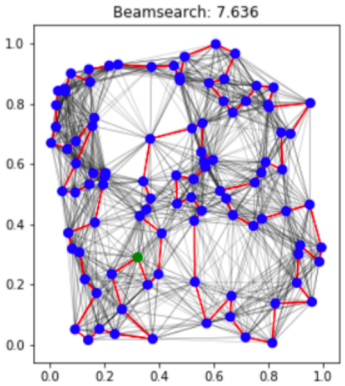




TSP100

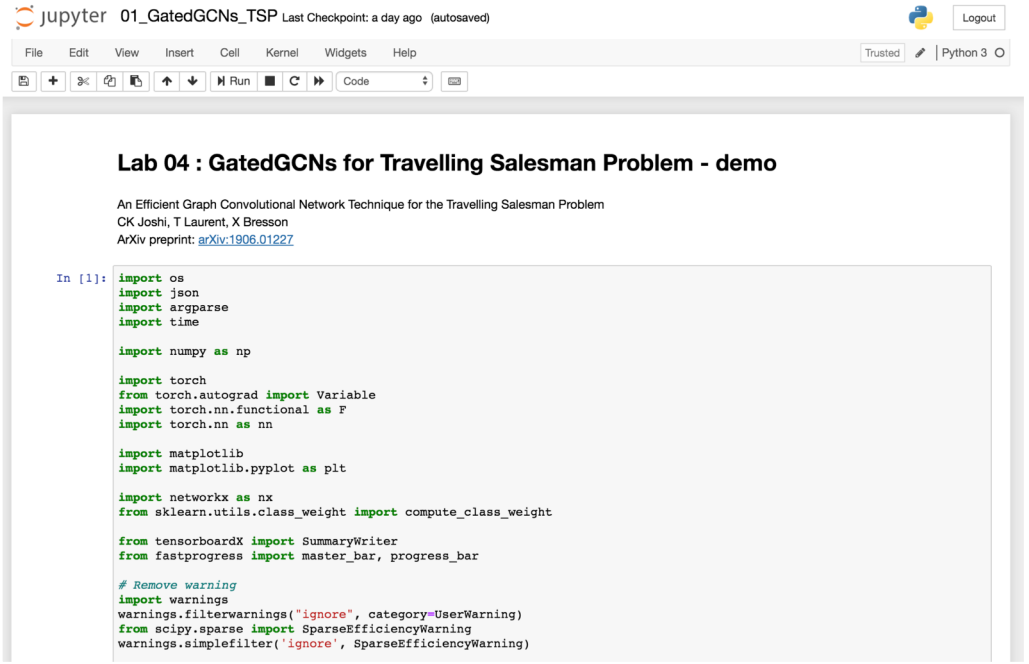






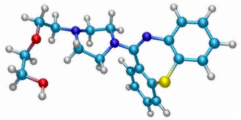
Lab 04

- GatedGCNs for Travelling Salesman Problem



Outline

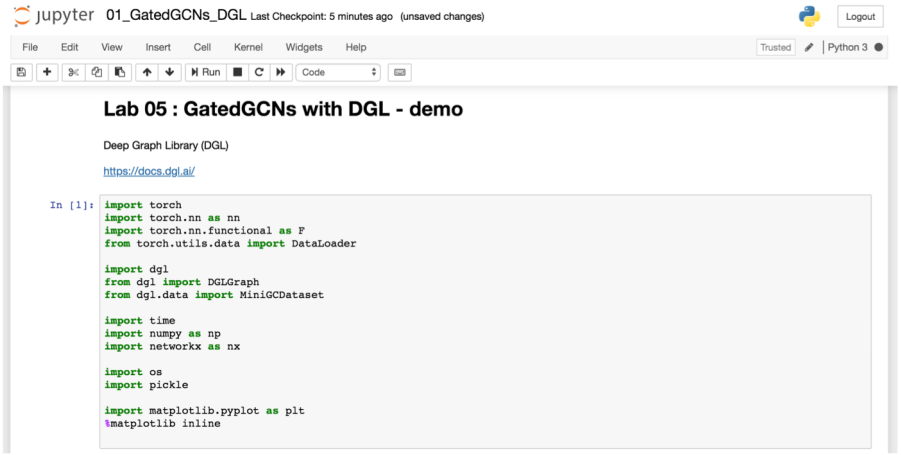
- **Part 3: Spatial Graph ConvNets**

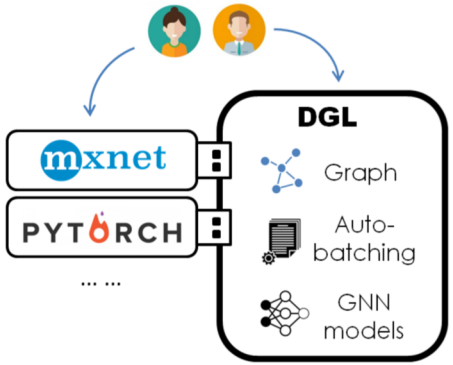


  - Problem Setting
  - Spatial Graph Architectures
  - Quantum Chemistry
  - Operations Research
  - **DGL**

Lab 05

- GatedGCNs with DGL (Deep Graph Library), NYU
  https://docs.dgl.ai





Lab 05

$\frac{1}{\sqrt{V_k}}, \frac{1}{\sqrt{E_k}}$ Normalization constants for better optimization

DGL creates a batch of graphs

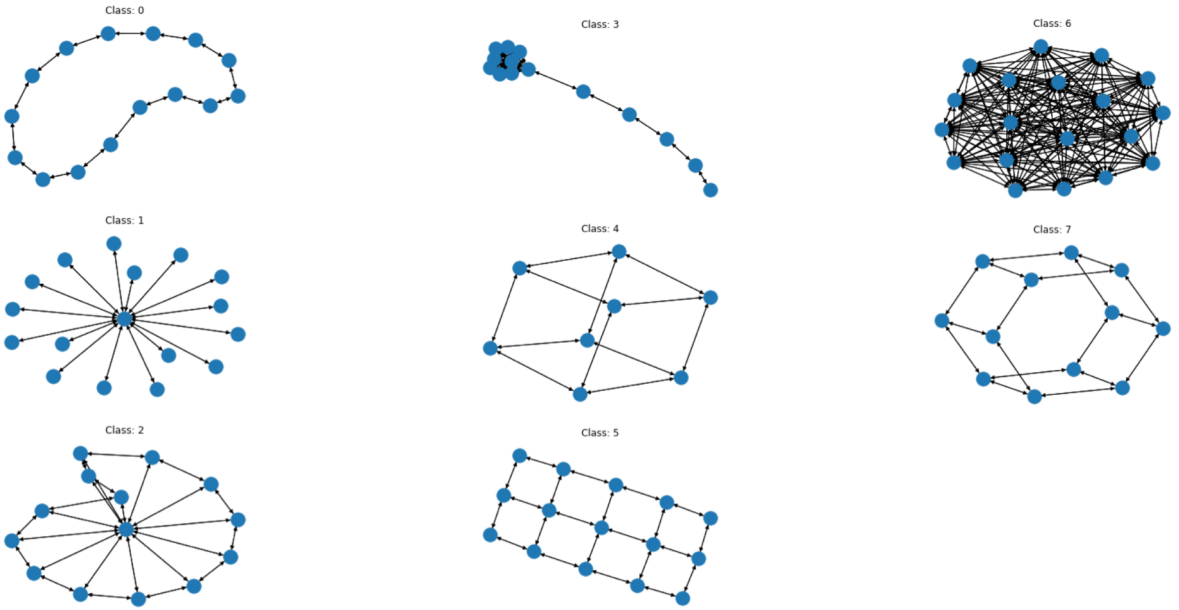
Create artificial node feature (input degree) and edge feature (value 1)

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


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

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

Lab 05

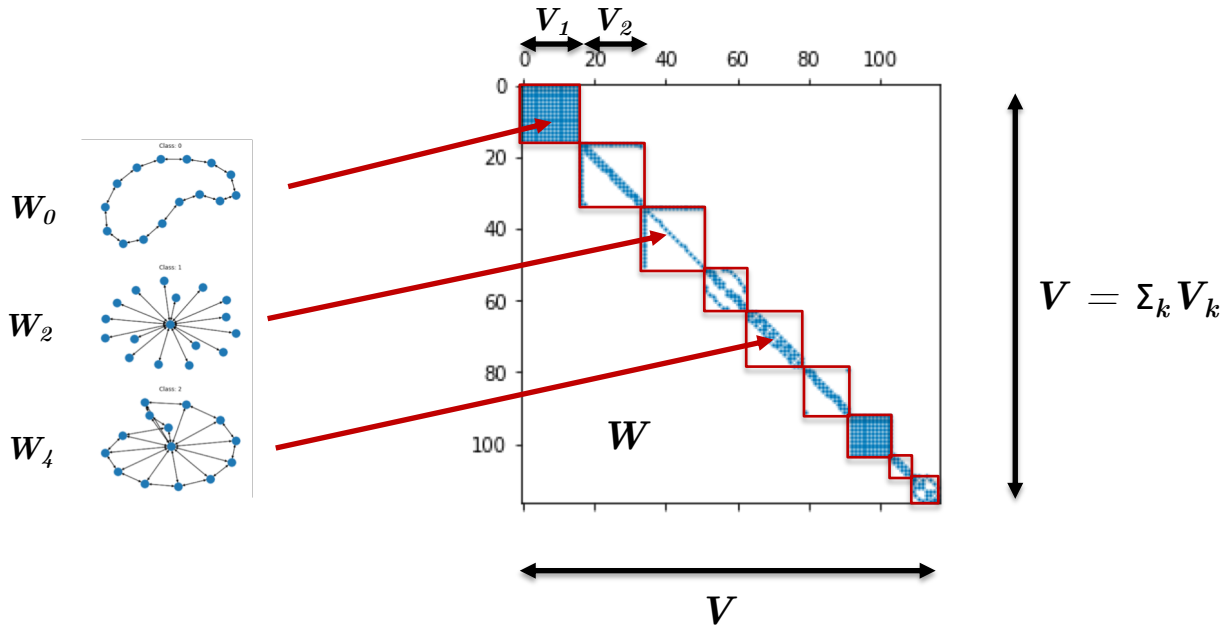
```
# use artifical graph dataset of DGL
trainset = MiniGCDataset(8, 10, 20)
```



Lab 05

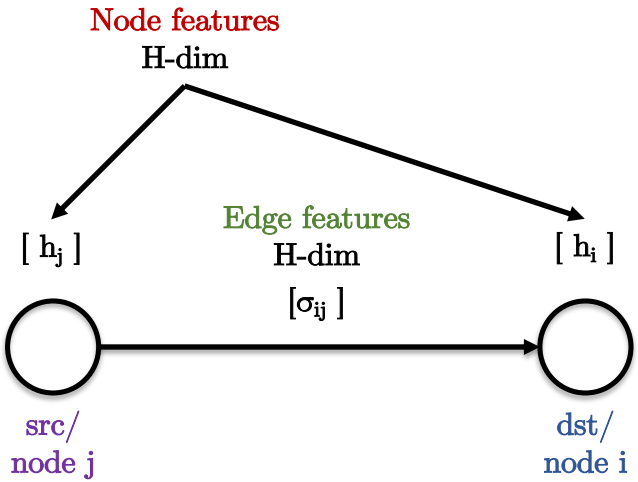
- Understanding DGL :
  - How to process K graphs of different lengths?

    Form a (big) sparse block diagonal matrix W with K adjacency matrices W_k .



Lab 05

- Basic structure of an edge in DGL :

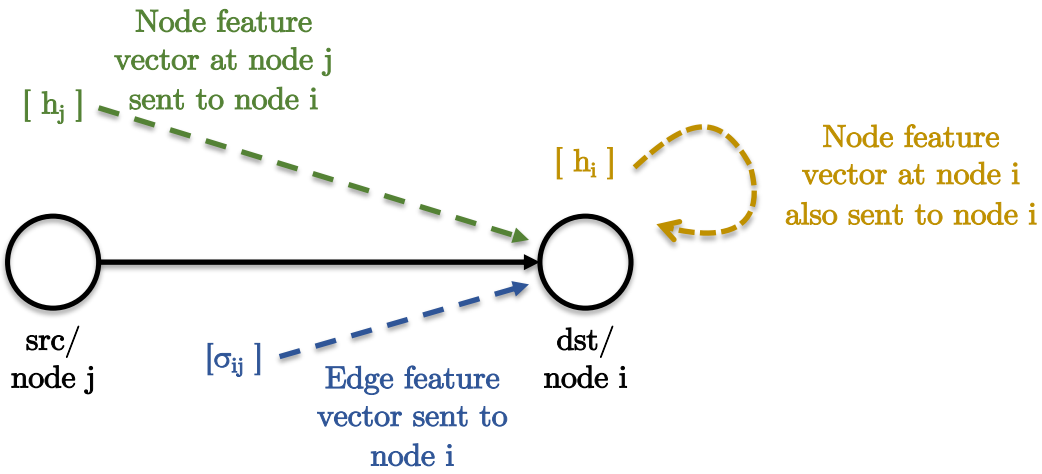


- Goal is to compute efficiently expressions of the form :

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

Lab 05

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



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

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

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

#edges

All message passing operations can be done in parallel !

Lab 05

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

#nodes

Sum over neighbors

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

#nodes in batch_1

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

#neighbors

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

$$h_i = \text{nodes.data}[\text{'}h\text{'}]\ .\ \text{size}() = V \text{ x } H$$

Lab 05

GatedGCN layer :

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        return h, e
```

Lab 05

MLP classifier layer

```python
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

```python
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 05

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

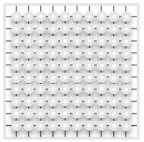
Train function

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

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

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

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

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

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

Main function

Outline

- Part 1: Traditional ConvNets



- Part 2: Spectral Graph ConvNets

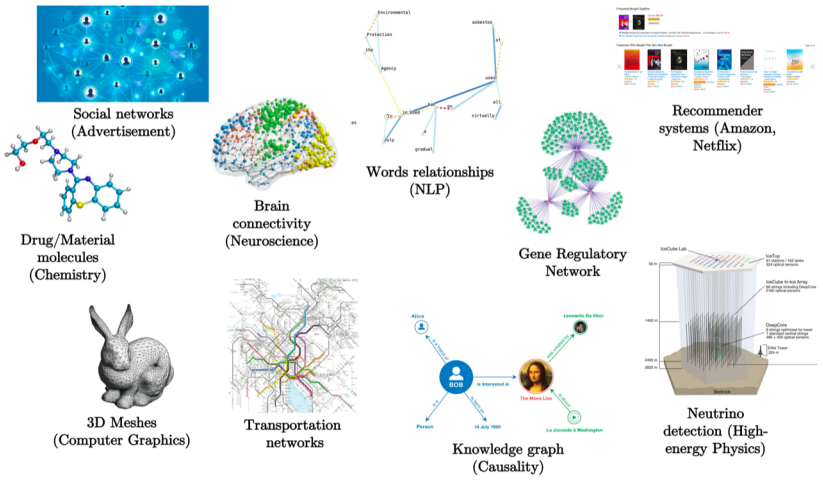
- Part 3: Spatial Graph ConvNets

- **Conclusion**

Conclusion

- Contributions :
  - Generalization of ConvNets to data on graphs
  - Re-design compositional property 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 :



Tutorials on Graph Deep Learning

NeurIPS'17
1,000-2,000 participants

CVPR'17
500-1,000 participants

SIAM Annual Meeting'18

- Organizers :
  - Y. LeCun (NYU, Chief AI Scientist at Facebook)
  - M. Bronstein (Imperial, Twitter)
  - J. Bruna (NYU)
  - A. Szlam (Facebook AI Research)
  - X. Bresson (NTU, Singapore)

2018 IPAM-UCLA Workshop

Organizing committee:

- Y. LeCun (NYU, Chief AI Scientist at Facebook)
- M. Bronstein (Imperial, Twitter)
- J. Bruna (NYU)
- A. Szlam (Facebook AI Research)
- S. Osher (UCLA)
- X. Bresson (NTU, Singapore)

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

Speaker List

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

2019 IPAM-UCLA Workshop

Organizing committee:

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

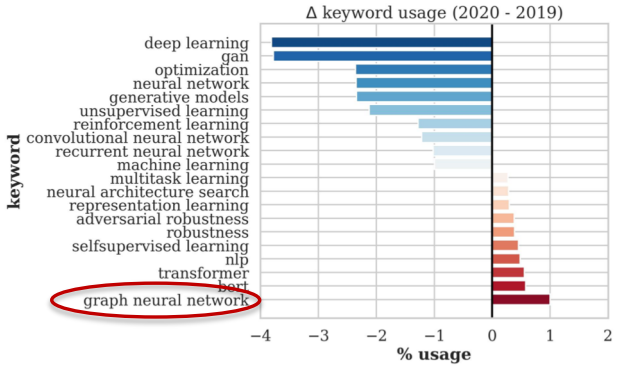
Video talks : https://bit.ly/2w8EtLV

Speaker List

Mikhail Belkin (Ohio State University)
Xavier Bresson (Nanyang Technological University, Singapore)
Soumith Chintala (Facebook AI Research)
Taco Cohen (Qualcomm AI Research)
Kostas Daniilidis (University of Pennsylvania)
Tom Goldstein (University of Maryland)
Bahram Jalali (University of California, Los Angeles (UCLA))
Thomas Kipf (Universiteit van Amsterdam)
Roy Lederman (Yale University, Applied Mathematics)
Jure Leskovec (Stanford University)
Federico Monti (Universita 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)

Graph Deep Learning

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



ICLR'20

Questions?