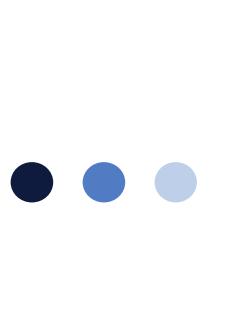




Point Cloud Processing II

Deep Learning methods

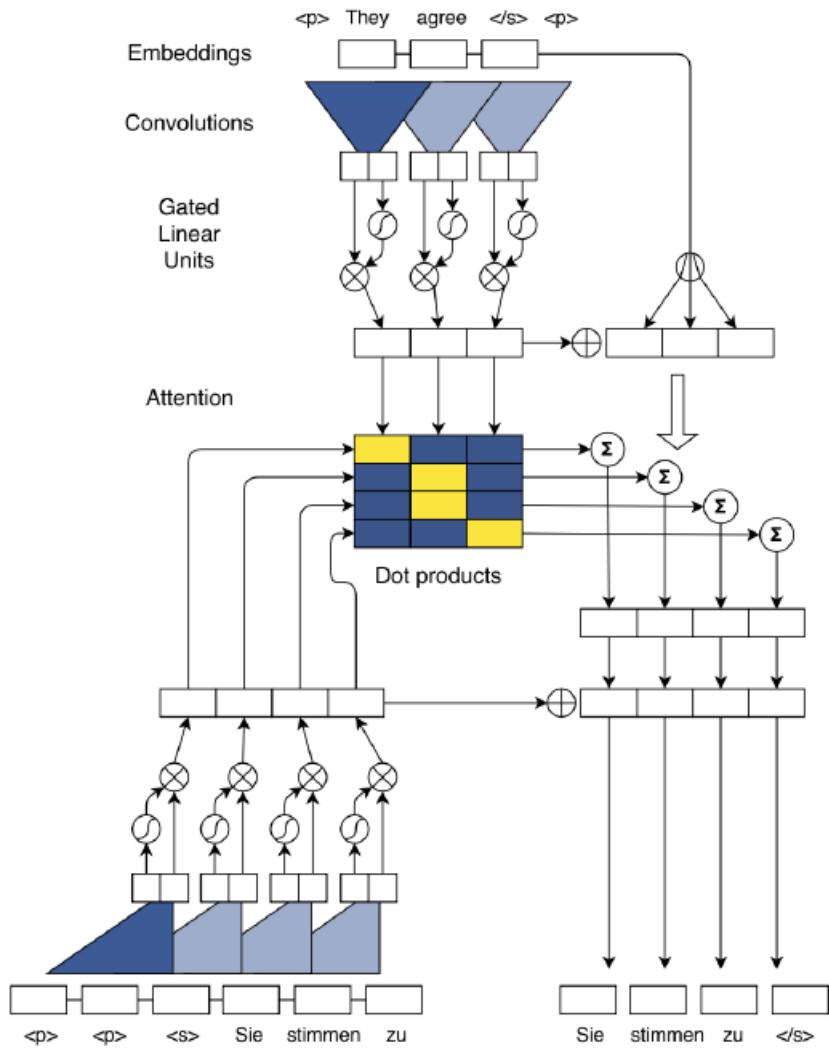
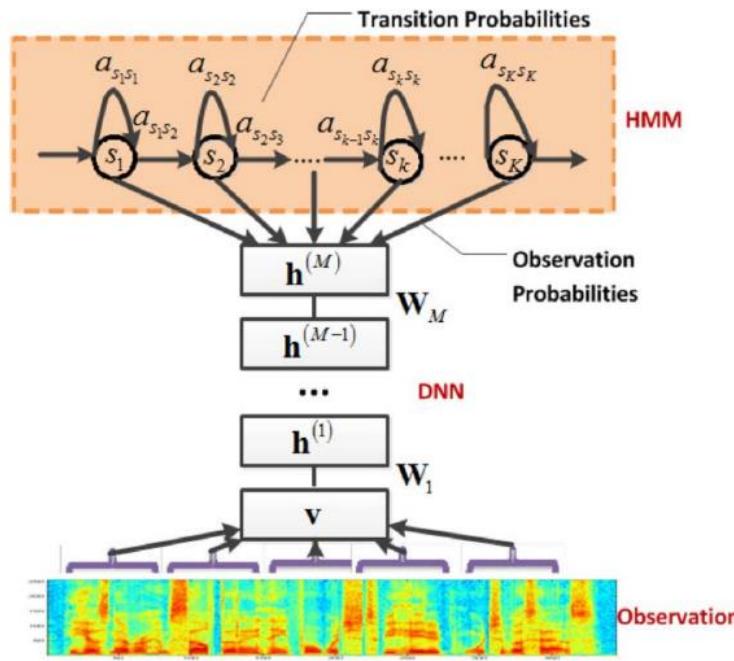


Why Deep Learning ?

- Deep learning has helped to boost performance in multiple fields
 - Speech recognition
 - Language translation
 - Image processing
 - ...
- In most cases,
 - Data were structured in regular grids
 - Parameter sharing was advantageous
- How can we use DL on point clouds?
 - More generally, on graphs?

Speech recognition & Language translation

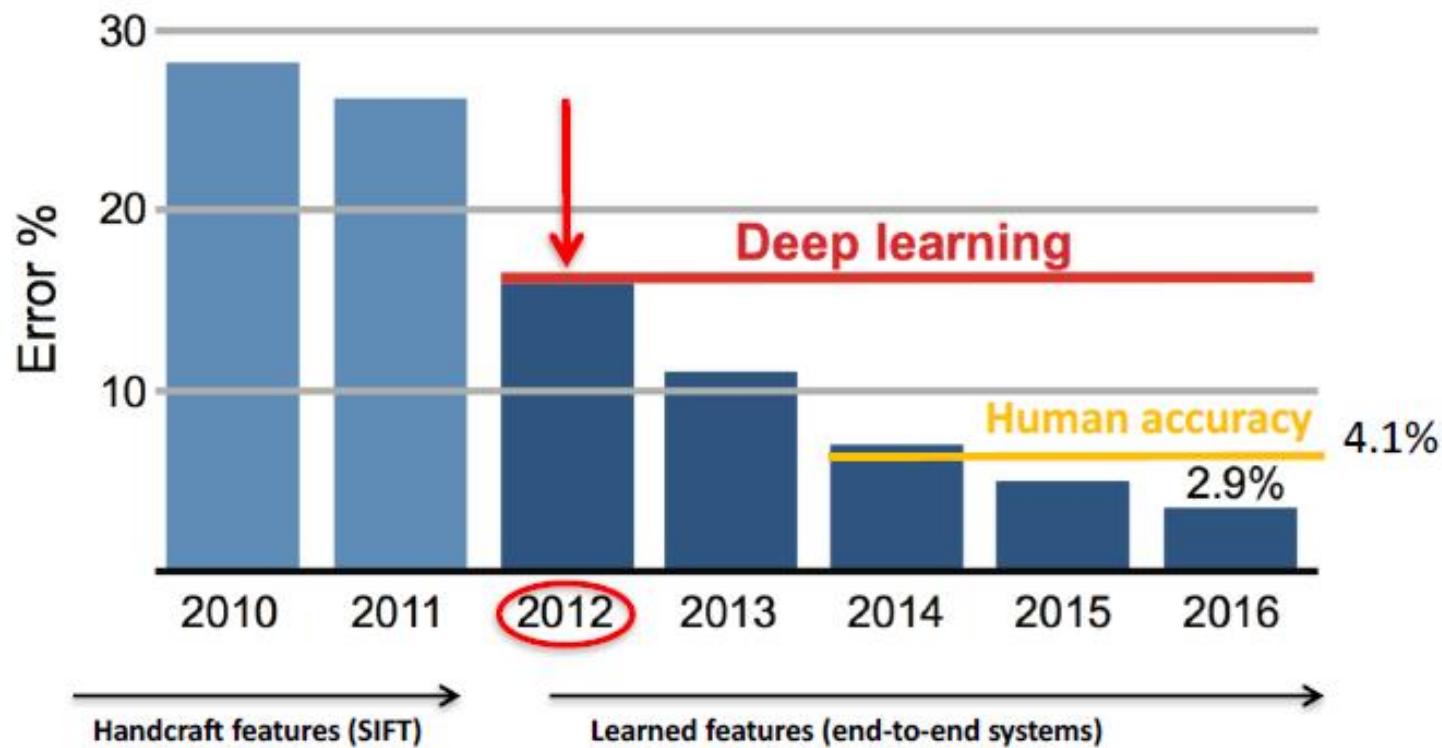
- Recurrent Neural Networks (RNNs)



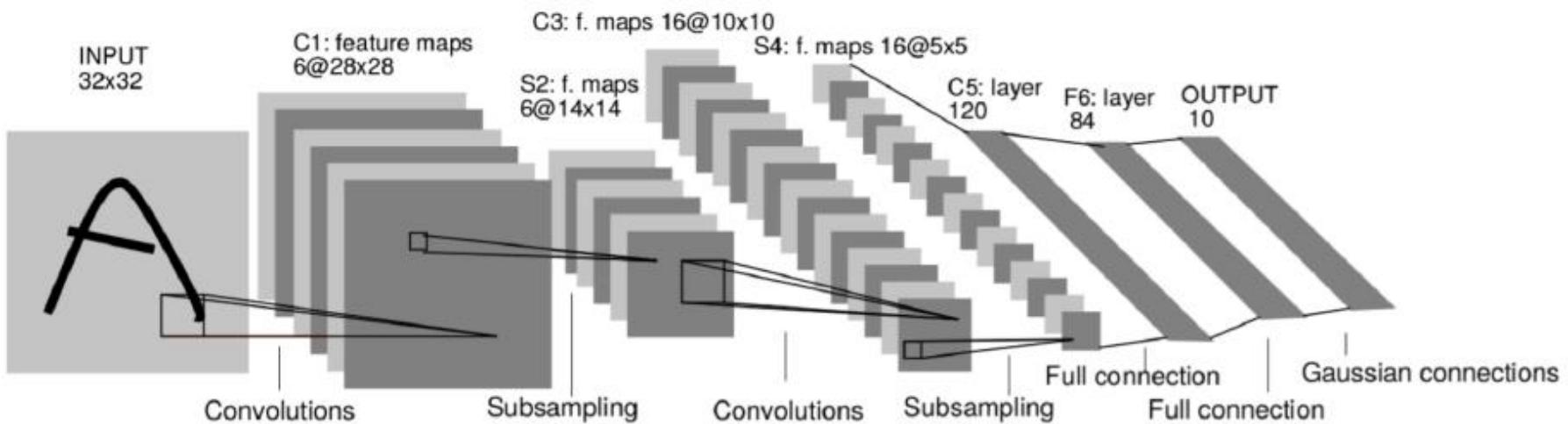
Deep learning w/images

- Convolutional Neural Networks (CNNs)

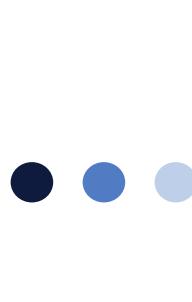
IMAGENET



Convolutional Neural Networks (CNN)



- Coarse of dimensionality
 - Images of $\sim 10^6$ pixels
 - Very small ratio samples / dimension
 - CNNs useful for high-dimensional problems
 - Locality / Parameter sharing



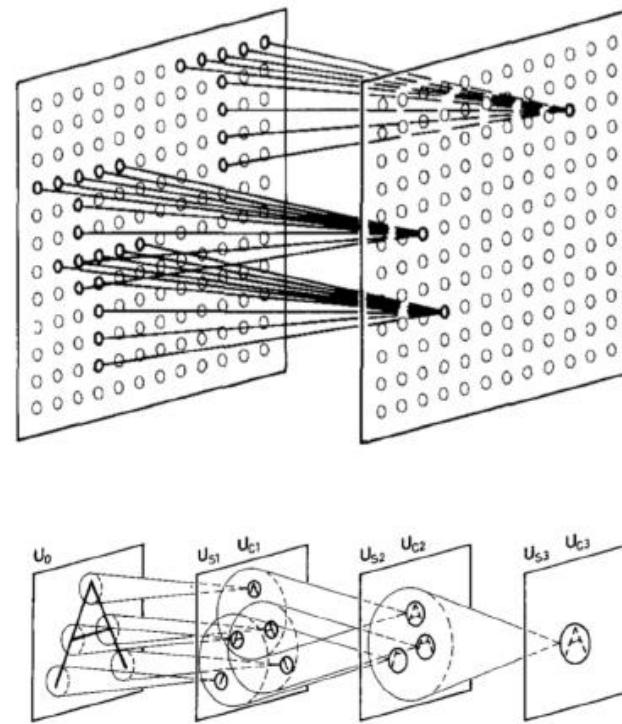
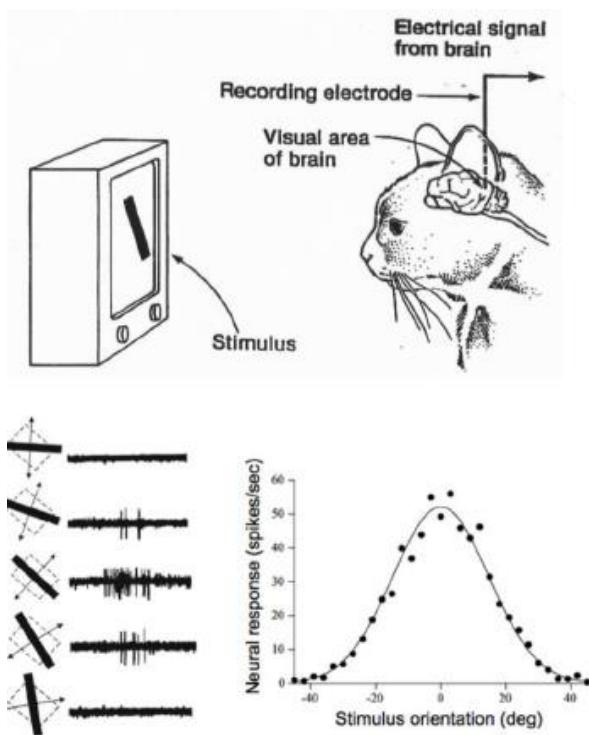
Compositional assumption

- Data (images, videos, sounds) are compositional
 - They are formed of patterns that are
 - Local
 - Stationary
 - Multi-scale (hierarchical)
- Convolutional neural networks
 - Leverage the compositionality structure of data
 - They extract compositional features

- Local
- Stationary
- Multi-scale (hierarchical)

Locality

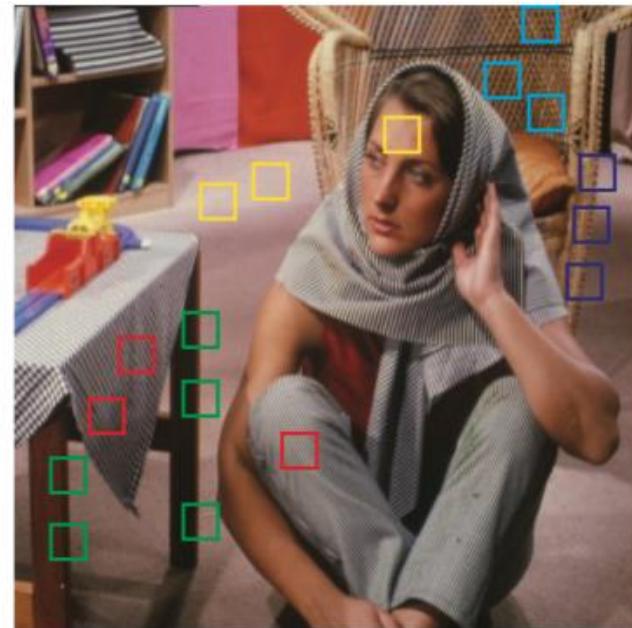
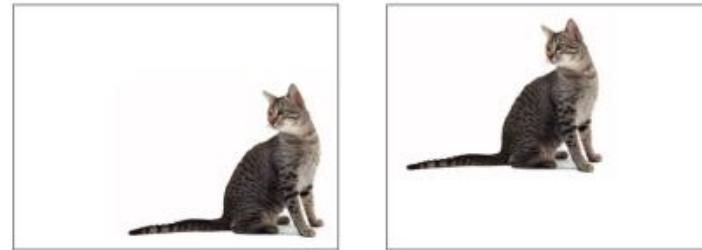
- Inspired in the visual cortex
- Local receptive fields
 - Activate in the presence of local features



Stationarity

- Local
- Stationary
- Multi-scale (hierarchical)

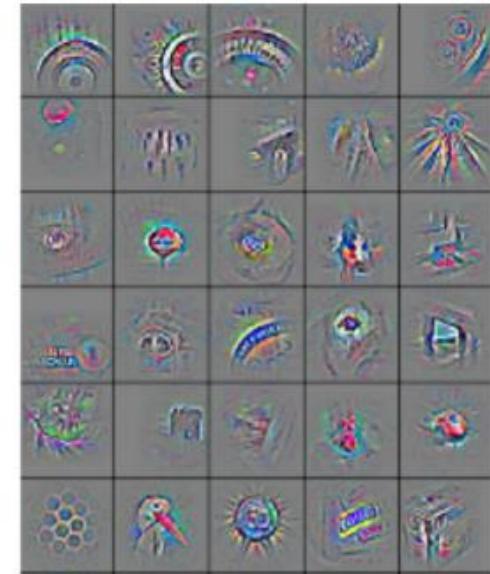
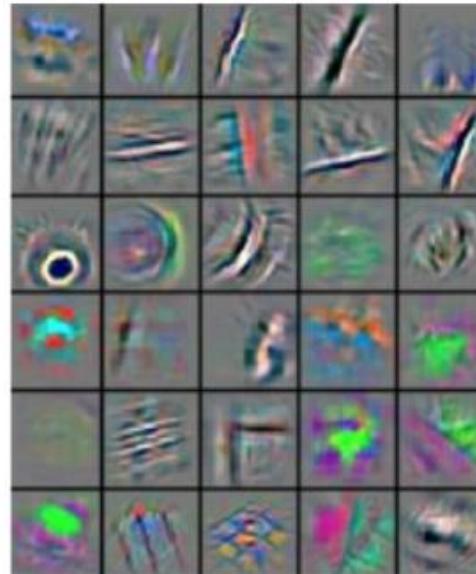
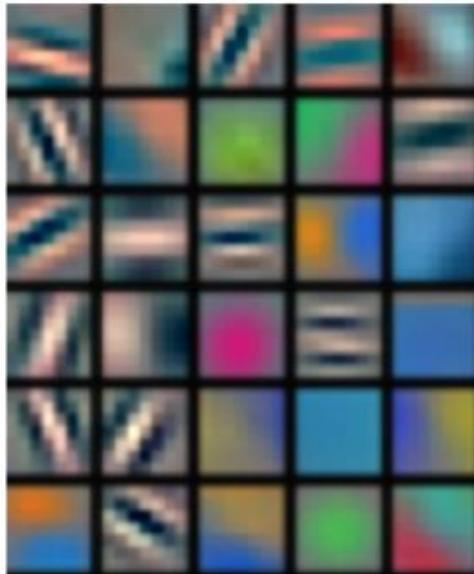
- Global stationarity
 - Translation invariance
- Local stationarity
 - Similar patches are shared across the data domain
 - Local invariance
 - Good for intra-class variation



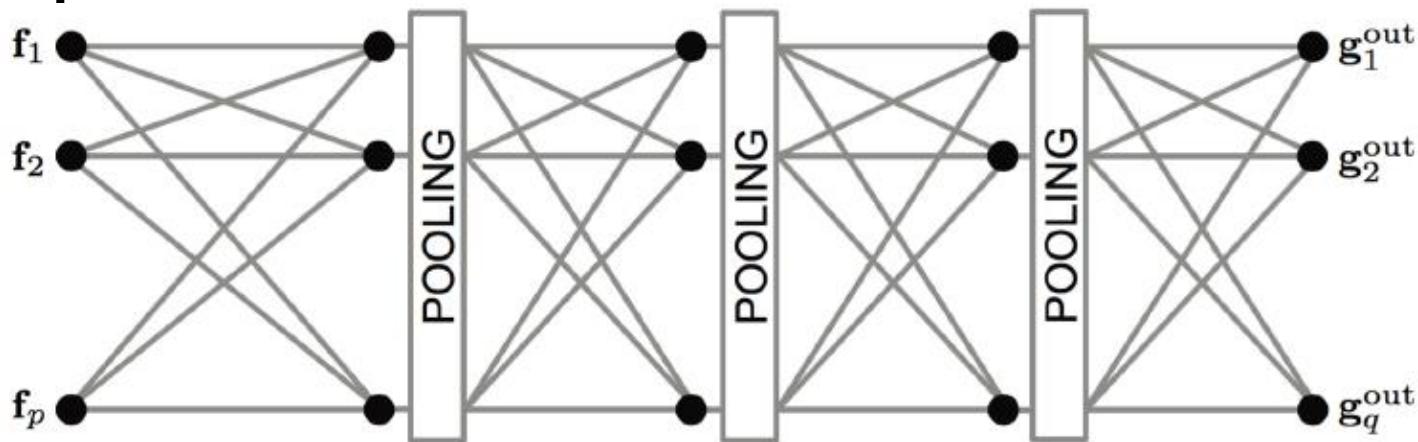
- Local
- Stationary
- Multi-scale (hierarchical)

Multi-scale

- Simple structures can be combined
 - To compose more abstract features
 - Those can be re-combined again in a similar fashion
 - Inspired by the brain
 - Visual primary cortex



Compositional layers



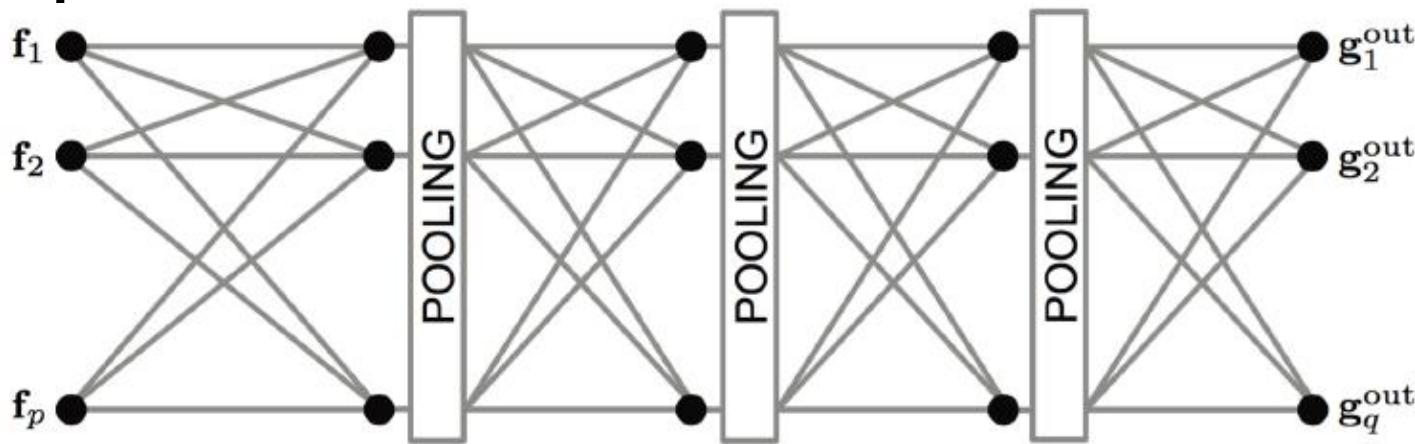
$$\begin{aligned} \mathbf{f}_l &= l\text{-th image feature (R,G,B channels)}, \dim(\mathbf{f}_l) = n \times 1 \\ \mathbf{g}_l^{(k)} &= l\text{-th feature map}, \dim(\mathbf{g}_l^{(k)}) = n_l^{(k)} \times 1 \end{aligned}$$

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

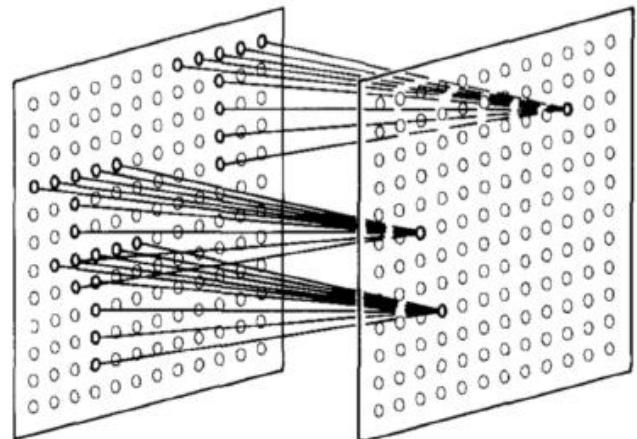
Activation, e.g. $\xi(x) = \max\{x, 0\}$ rectified linear unit (ReLU)

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

Compositional layers



- Convolutional layers
 - Parameters are shared across different “neurons”
 - Huge reduction in computational complexity
 - With respect to Fully-Connected (FC) layers



Data domain for CNNs

- Domains with regular sampling structures
- 1-D Euclidean domain
 - Sentences, words
 - Sound
- 2-D Euclidean domain
 - Images
- 3-D Euclidean domain
 - Video
 - Volumes

O ROMEO, WHEREFORE ART THOU ROMEO?
Although we use "wherefore" if at all, as an synonym for "why," Juliet uses the word in a more limited sense. By "wherefore" Juliet means "for what purpose?" If she had merely asked "Why art thou Romeo?" she would have been asking for the reason of his being there. But "why" — "why" — "wherefore" — "what cause" (in the first and "for what purpose" in the last), "wherefore" clearly emphasizes the latter sense, which is why "why" and "wherefore" are different things.

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

WHAT'S IN A NAME? THAT WHICH WE CALL A BIRD IS NOT

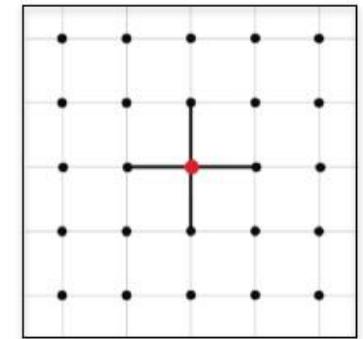
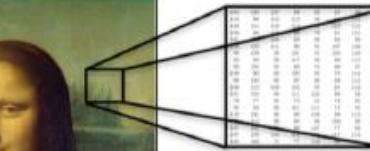
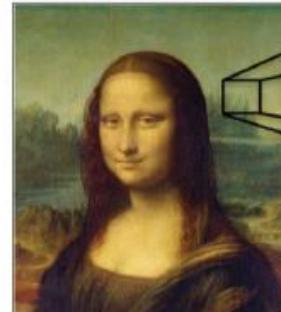
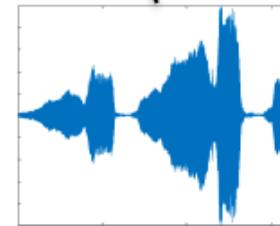
SWARMS OF WORMS. SHAKESPEARE TODAY THIS IS IT: Both "swear" one instant bird, although the fitter is, as every Kestrel, merely a passerby. From the variously dissociation-to-the-ones advertisements, these phrases have served generations with complete

"What's in a name?" is the less specific of the two phrases, and also the less common. Julian has many references in a difference from the ones of "What's a Interlogue," moving, like a good Shakespeare student, from the particular to the general. Names in general, of course, ought to be specific, and Julian's choice of "bird" is a good example of this. It's a little difficult to understand, though. Shakespeare knew that he old have to fit out a line and a half of blank verse. Regarding Juliet's use of "word" instead of "name," we can perhaps be grateful; one already uses "name." So names is a better evasive base.

"That which we call a rose by any other name would smell as sweet" seems broad to the modern ear. But we're accustomed to the general idea of names, and Julian's use of "name" is a good example. We're a little difficult to understand, though. Shakespeare knew that he old have to fit out a line and a half of blank verse. Regarding Juliet's use of "word" instead of "name," we can perhaps be grateful; one already uses "name." So names is a better evasive base.



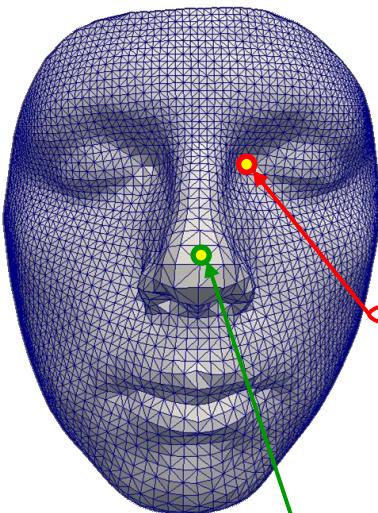
1D grid



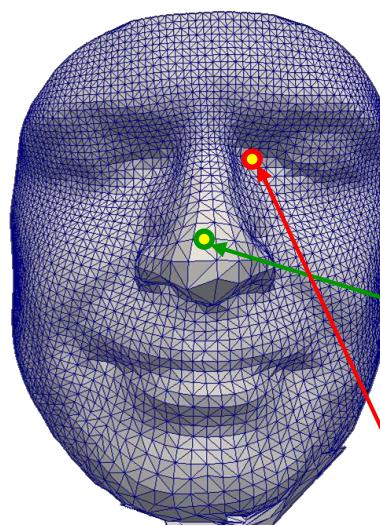
2D grids

How about point clouds?

Input points are inherently not ordered



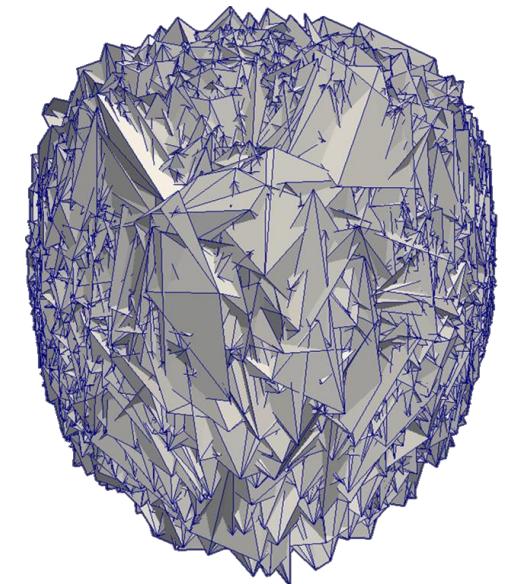
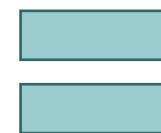
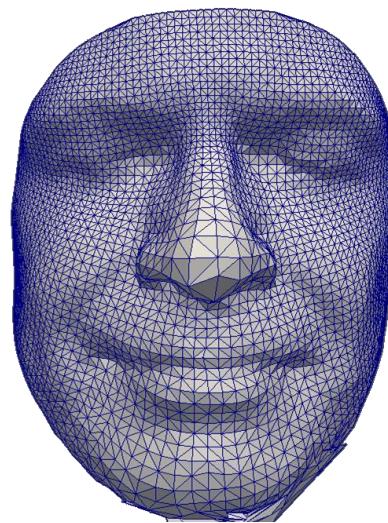
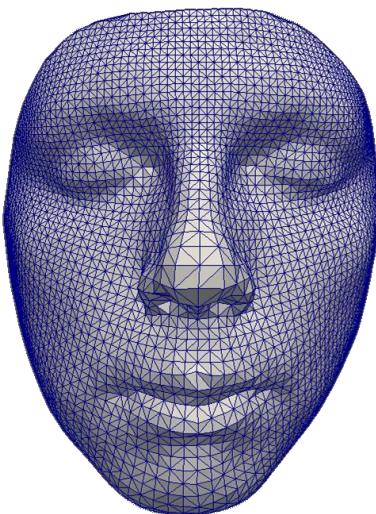
```
ply
format ascii 1.0
element vertex 50140
property float x
property float y
property float z
element face 99698
property list uchar int vertex_index
end_header
3.060000 -98.770000 -141.990000
4.440000 -105.300000 -135.780000
9.430000 -104.880000 -139.440000
10.770000 -104.740000 -137.530000
10.400000 -108.510000 -138.260000
10.520000 -106.680000 -137.630000
8.570000 -98.240000 -142.080000
10.410000 -98.130000 -140.020000
10.330000 -97.290000 -140.040000
9.890000 -96.610000 -137.690000
9.960000 -98.340000 -137.080000
9.880000 -98.140000 -136.340000
10.200000 -95.580000 -140.100000
10.230000 -93.930000 -140.330000
9.820000 -94.990000 -138.350000
9.680000 -96.190000 -135.970000
9.550000 -95.620000 -133.700000
9.770000 -97.880000 -135.490000
9.630000 -93.270000 -138.780000
9.340000 -92.920000 -136.730000
9.290000 -93.270000 -135.460000
9.690000 -95.640000 -137.120000
9.510000 -94.540000 -136.370000
9.400000 -93.950000 -134.440000
9.310000 -93.260000 -132.070000
10.730000 -93.320000 -141.410000
9.460000 -88.070000 -140.420000
10.060000 -90.110000 -141.020000
9.490000 -87.160000 -140.250000
8.760000 -86.630000 -137.330000
9.810000 -89.620000 -140.580000
8.700000 -82.260000 -139.540000
9.050000 -85.310000 -140.120000
8.870000 -82.470000 -142.240000
```



```
ply
format ascii 1.0
element vertex 82938
property float x
property float y
property float z
element face 165137
property list uchar int vertex_index
end_header
-59.760000 -36.120000 -149.910000
-60.070000 -36.140000 -145.260000
-59.150000 -37.290000 -151.650000
-59.460000 -37.310000 -150.300000
-60.430000 -33.580000 -148.200000
-59.580000 -34.810000 -151.750000
-59.910000 -34.880000 -150.250000
-60.270000 -34.890000 -148.310000
-59.470000 -36.070000 -151.490000
-60.830000 -32.330000 -145.950000
-60.970000 -32.290000 -144.340000
-60.680000 -33.610000 -146.530000
-60.540000 -34.880000 -146.580000
-55.530000 -69.120000 -152.370000
-55.410000 -69.300000 -152.810000
-55.950000 -67.920000 -150.810000
-55.720000 -68.250000 -151.860000
-55.380000 -68.420000 -153.080000
-55.620000 -68.520000 -152.160000
-55.650000 -68.870000 -151.960000
-55.390000 -68.840000 -152.980000
-56.090000 -67.270000 -150.110000
-56.270000 -64.760000 -149.950000
-56.240000 -65.980000 -149.530000
-55.740000 -67.810000 -151.860000
-56.170000 -64.160000 -151.020000
-56.310000 -64.280000 -150.060000
-56.480000 -64.330000 -148.940000
-56.430000 -64.830000 -148.850000
-56.310000 -65.360000 -149.210000
-56.740000 -63.710000 -147.800000
-56.710000 -64.240000 -147.600000
-56.520000 -63.790000 -149.100000
-56.800000 -63.110000 -147.720000
```

How about point clouds?

Input points are inherently not ordered



2

This does not work because surface points are
not in correspondence

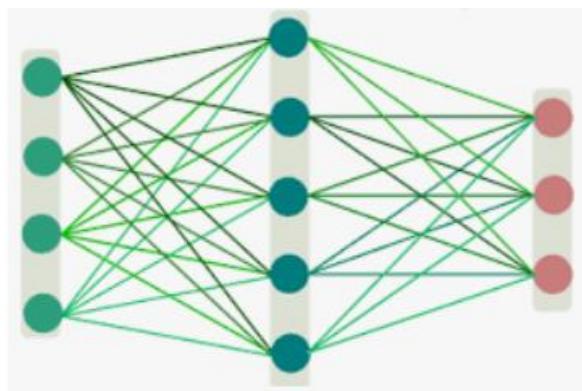
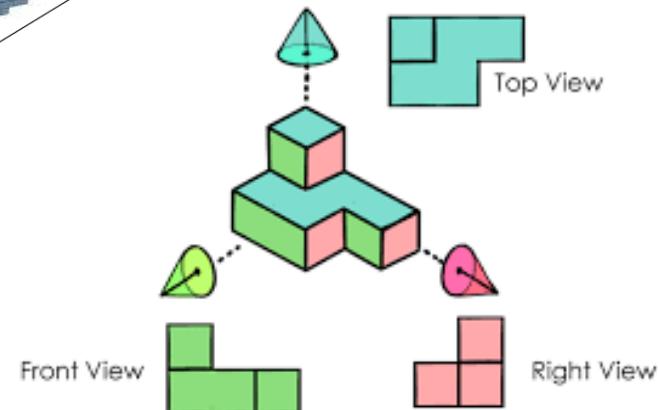
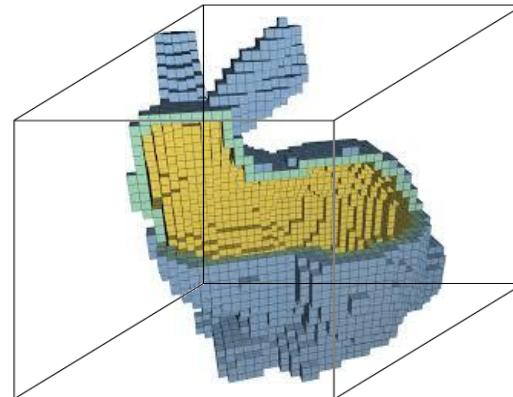


How can we apply DL to point clouds?

- Transform the representation
 - Voxelization
 - 3D-CNN
 - Projection / rendering
 - 2D-CNN
 - Feature extraction
 - Fully-connected networks
- Networks specifically designed for point clouds
 - PointNet / PointNet++
- Graph Neural Netowkrs
 - Generalization to non-Euclidean sampling grids

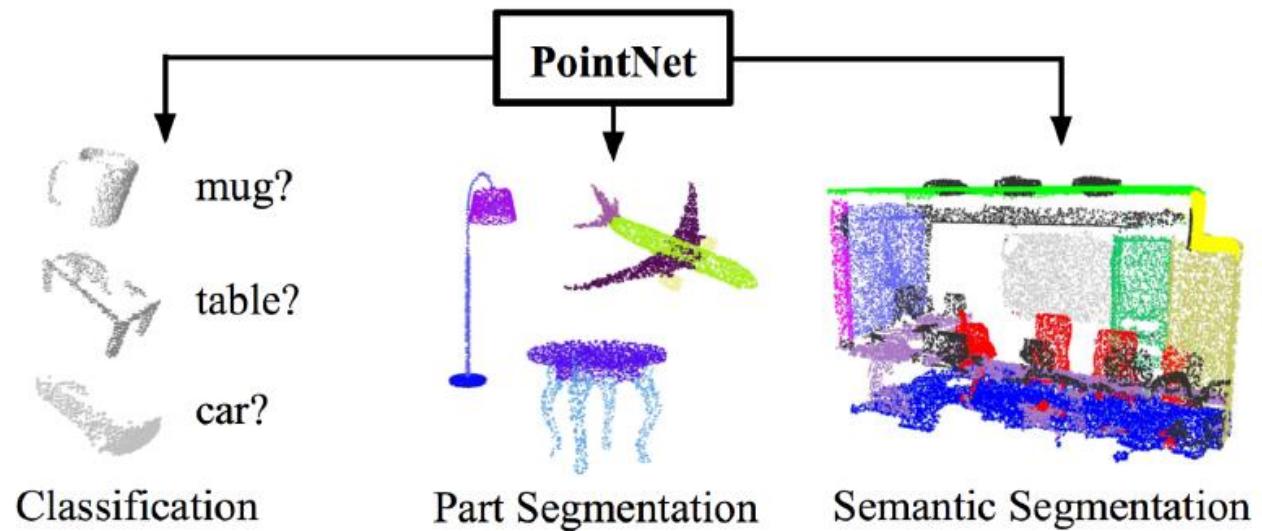
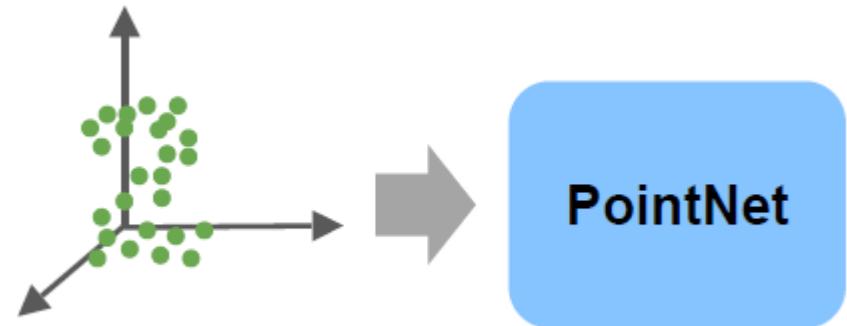
DL on point-clouds Transformed Representations

- Voxelization
 - 3D-CNN
- Projection / rendering
 - 2D-CNN
- Feature extraction
 - Fully-connected networks



••• DL on point-clouds: PointNet

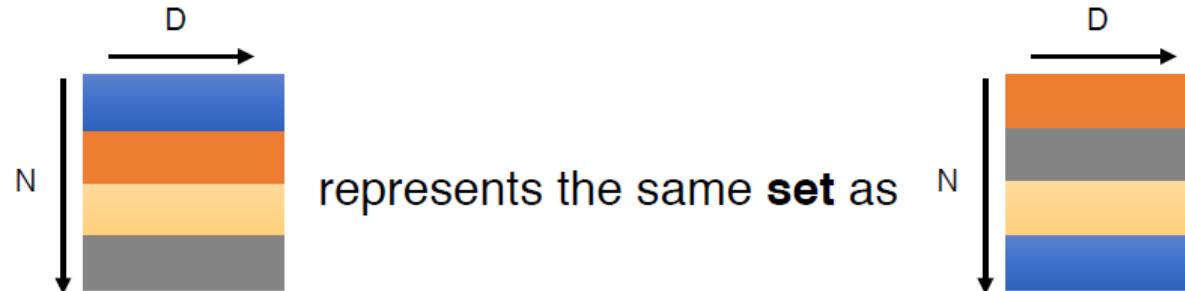
- End-to-end learning for point data
 - Scattered
 - Unordered
- Unified framework for various tasks



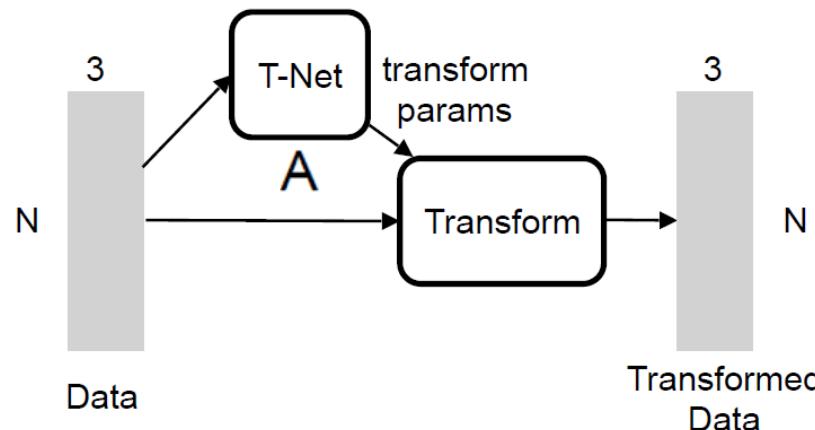
PointNet



- Unordered point set as input
 - The model should be invariant to permutations



- Invariance to rigid transformations



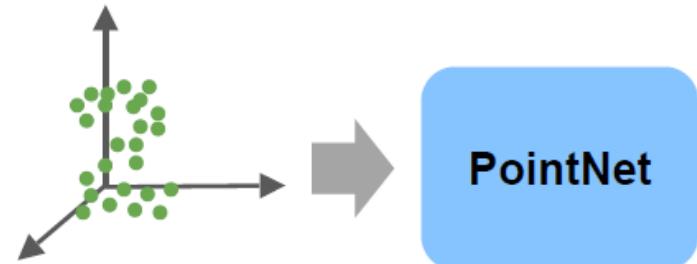
Regularization:

Transform matrix A close to orthogonal:

$$L_{reg} = \|I - AA^T\|_F^2$$



PointNet



- The model should be invariant to permutations

$$f(x_1, x_2, \dots, x_n) \equiv f(x_{\pi_1}, x_{\pi_2}, \dots, x_{\pi_n}), \quad x_i \in \mathbb{R}^D$$

- Examples

$$f(x_1, x_2, \dots, x_n) = \max\{x_1, x_2, \dots, x_n\}$$

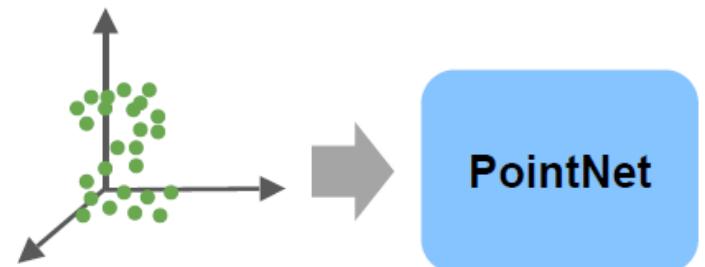
$$f(x_1, x_2, \dots, x_n) = x_1 + x_2 + \dots + x_n$$

...

- Then:
 - How can we construct a family of symmetric functions with neural networks?

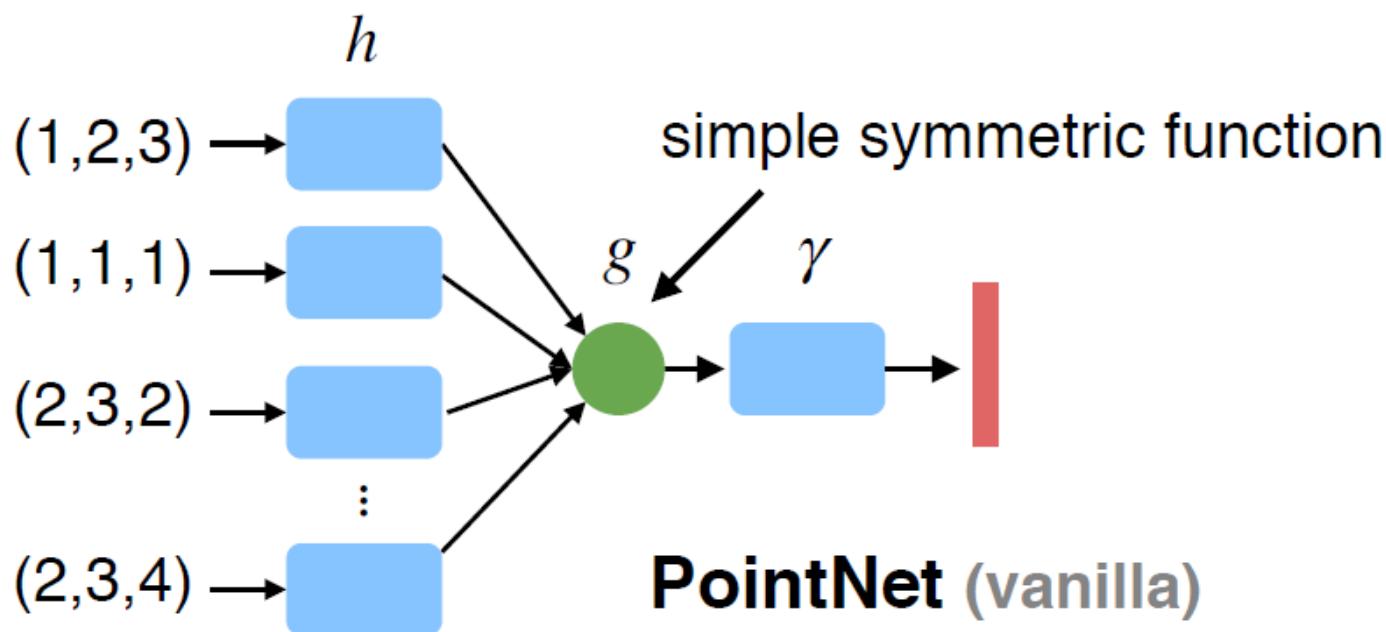


PointNet

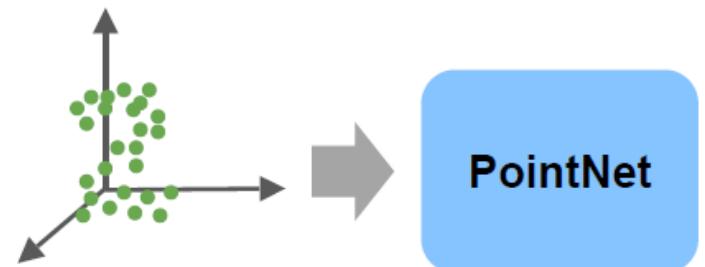


- For the function below
 - f is symmetric if g is symmetric

$$f(x_1, x_2, \dots, x_n) = \gamma \circ g(h(x_1), \dots, h(x_n))$$

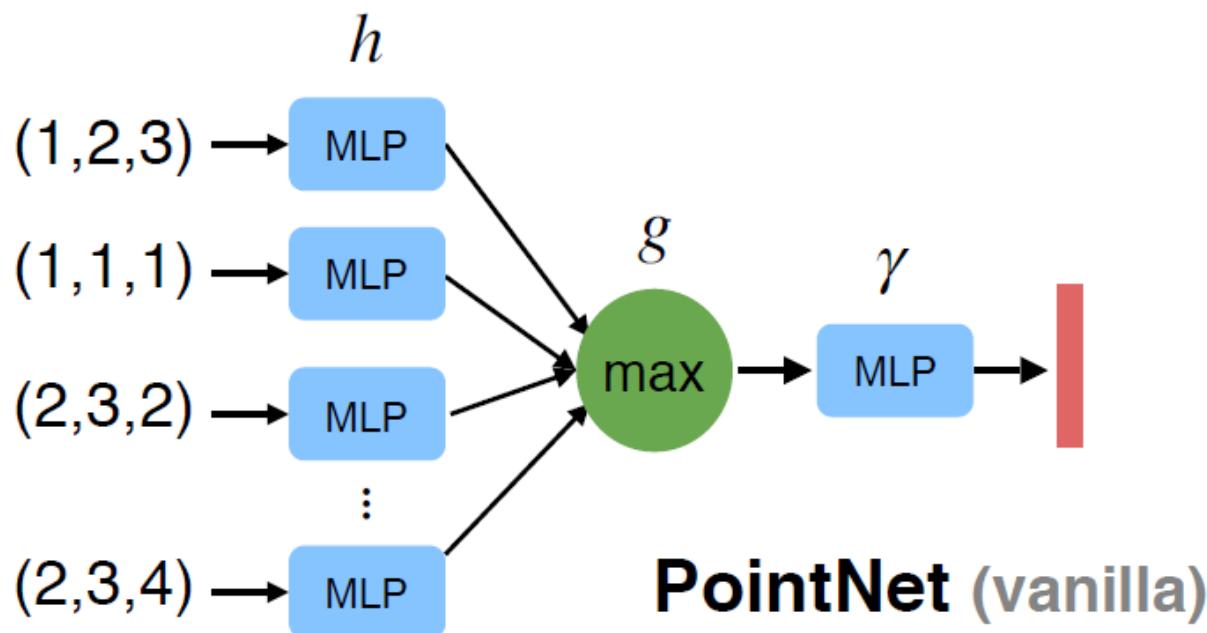


PointNet

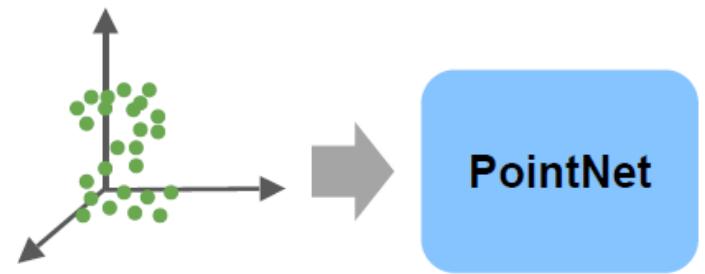


- In PointNet

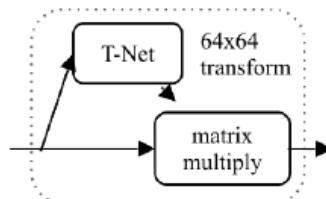
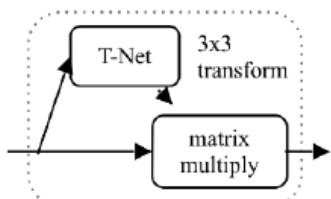
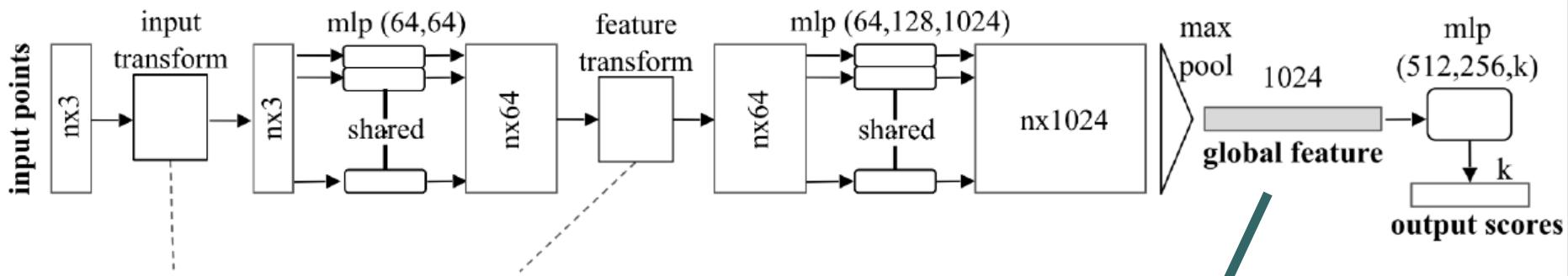
- g was chosen to be the MAX function
- γ was chosen to be a Multi-Layer Perceptron (MLP)



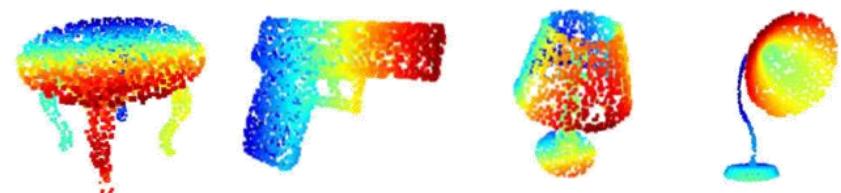
PointNet



PointNet architecture



Original Shape:



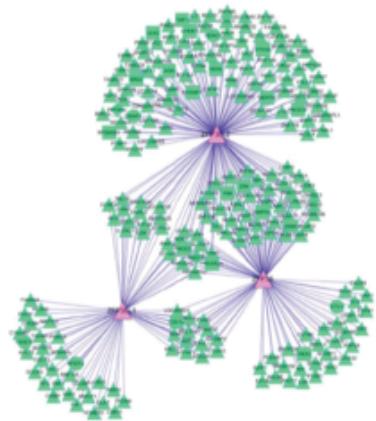
Critical Point Sets:



Non-Euclidean data

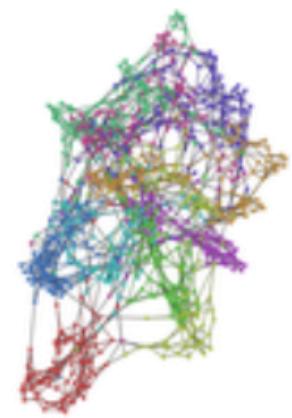


Social networks

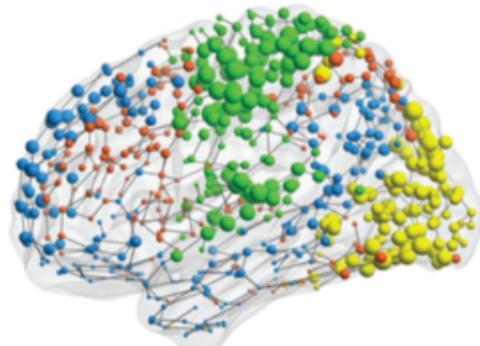


Regulatory networks

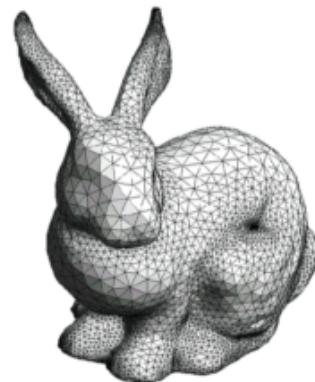
=



Graphs/
Networks



Functional networks



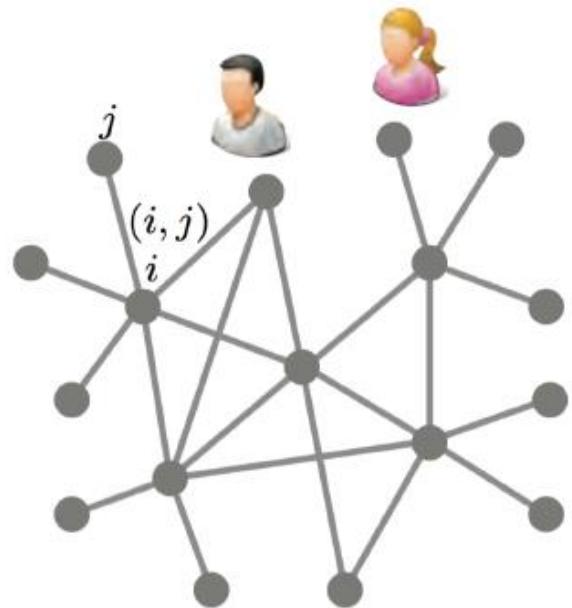
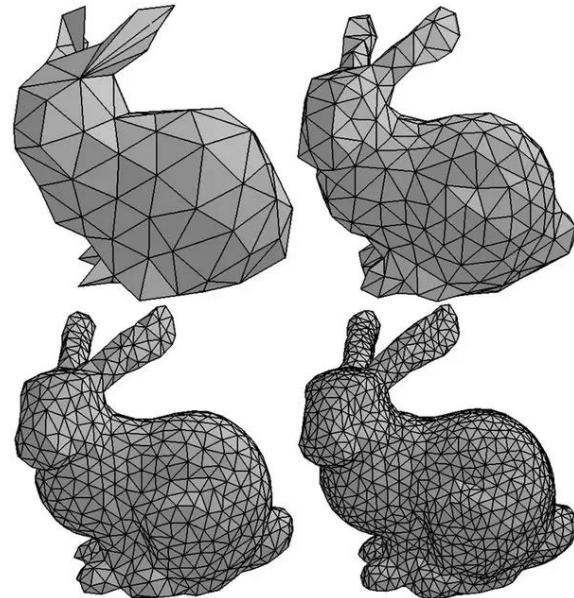
3D shapes



Non-Euclidean data

- Can be handled generically by using graphs

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





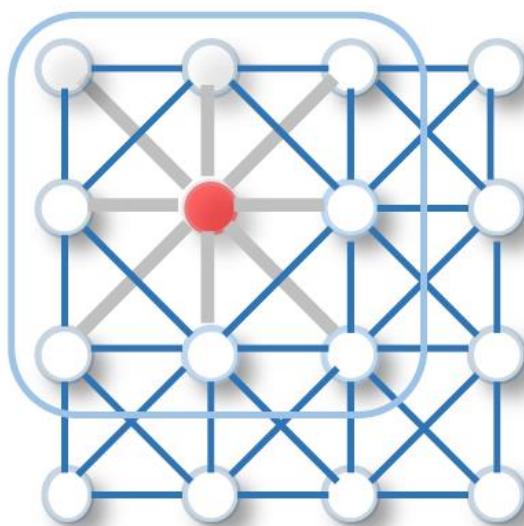
DL in Non-Euclidean data

- Assumptions
 - Non-Euclidean data are
 - Locally stationary
 - Compositional
- Challenges
 - How can we extend CNNs to graph-structure data?
 - How can we extend convolutions to graphs?
 - How to define compositionality on graphs?
 - Convolution
 - Pooling
 - How can we make the above operations efficient?
 - Computation in graphs tends to be expensive

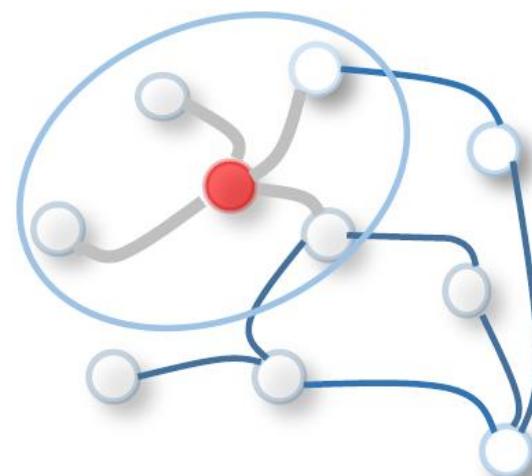


DL in Non-Euclidean data

- How about a direct extension based on neighborhoods?
 - Convolution / Pooling



Euclidean
neighborhood

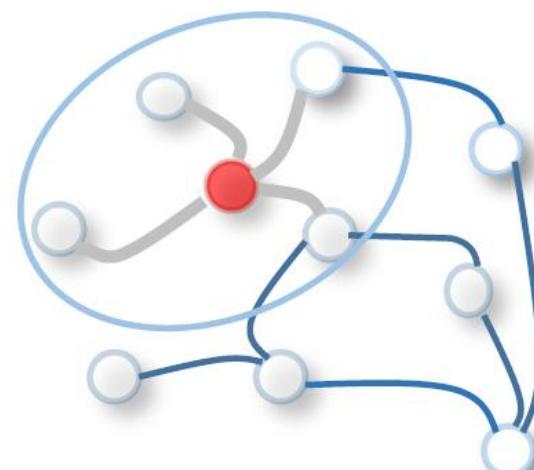


Non-Euclidean
neighborhood



DL in Non-Euclidean data

- How about a direct extension based on neighborhoods?
 - Convolution / Pooling
- Numerous disadvantages
 - Non-regular neighborhoods
 - Variable sizes / cardinality
 - Cannot properly order points
 - Cannot share weights
 - Not compositional
 - Not efficient



Non-Euclidean
neighborhood

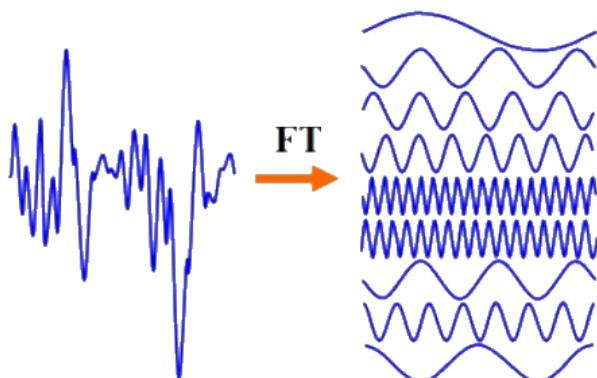


DL in Non-Euclidean data

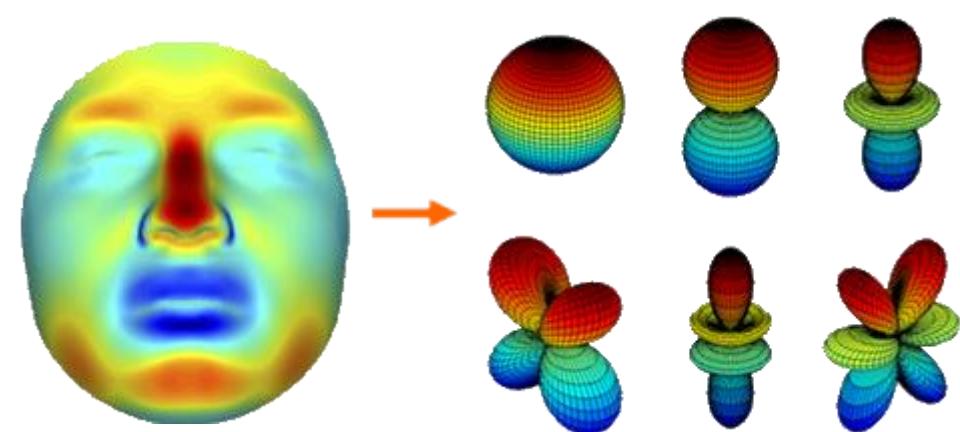
- How about a direct extension based on neighborhoods?
 - Convolution / Pooling
- 3 main solutions have been proposed
 - DL extensions based on RNNs
 - Message passing between neighboring nodes
 - DL extensions based on CNNs
 - Convolutions in the spectral domain
 - Convolutions directly on the graph

Spectral methods on graphs

- Relies on the decomposition of the surface geometry into its spatial frequency components,
 - Much like a Fourier transform for 1D signals



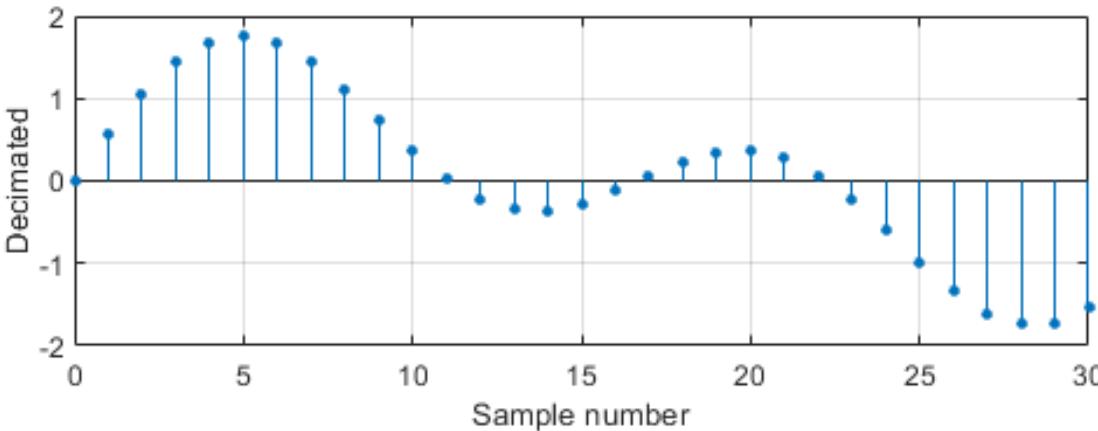
Signal in 1D



Surface in 3D

Spectral methods on graphs

- The spectrum from the Laplacian operator
 - Eigen-decomposition
 - Eigen-values are the spatial frequencies
 - Eigenvectors are the basis functions
 - Illustration from the 1-D case
 - The sampled function is defined over a uniform 1-D grid

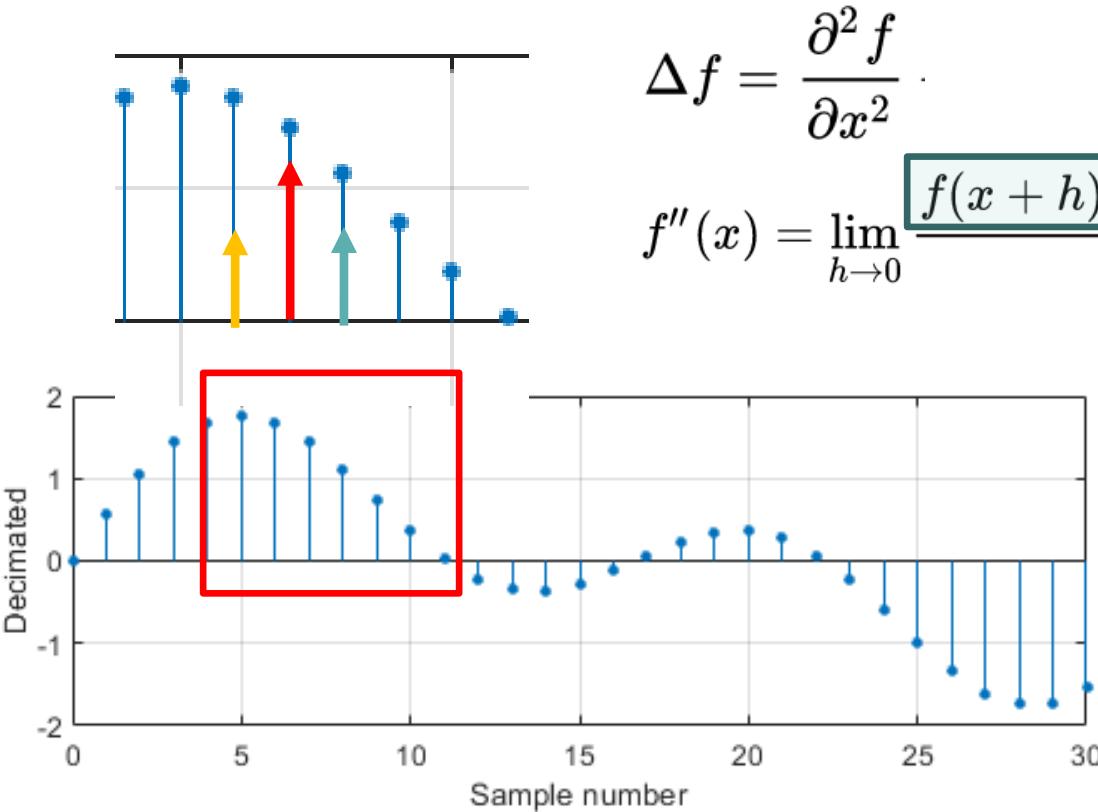


$$\Delta x = -K x$$

$$K = \begin{pmatrix} 2 & -1 & & & -1 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ -1 & & & -1 & 2 \end{pmatrix}$$

Spectral methods on graphs

- Illustration from the 1-D case
 - The sampled function is defined over a uniform 1-D grid



$$\Delta f = \frac{\partial^2 f}{\partial x^2}$$

$$f''(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

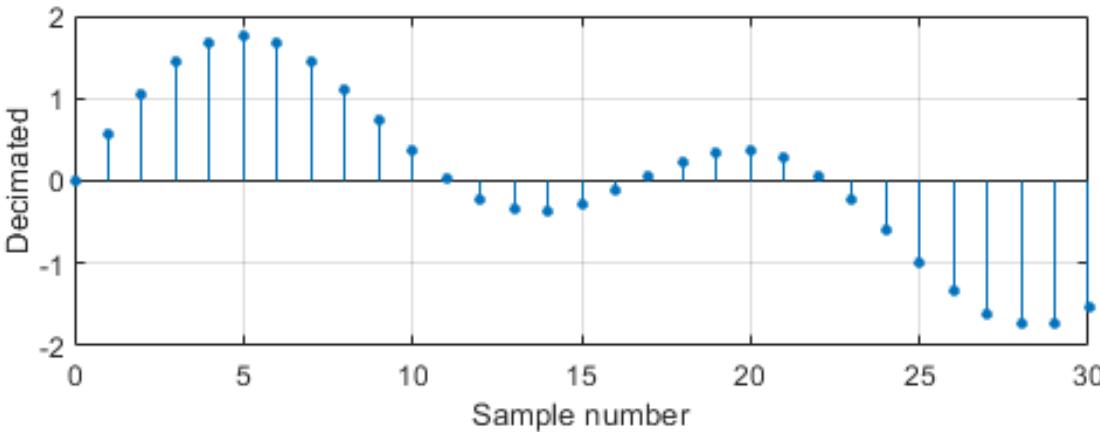
$$\Delta x = -K x$$

$$K = \begin{pmatrix} 2 & -1 & & & & -1 \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 \\ & & & -1 & -1 & 2 \end{pmatrix}$$

Spectral methods on graphs

- Illustration from the 1-D case
 - The sampled function is defined over a uniform 1-D grid
 - The eigendecomposition yields

$$\Delta \phi_j = \lambda_j \phi_j \quad \phi_j = \begin{cases} \sqrt{1/n} & \text{if } j = 1 \\ \sqrt{2/n} \sin(2\pi h \lfloor j/2 \rfloor / n) & \text{if } j \text{ is even} \\ \sqrt{2/n} \cos(2\pi h \lfloor j/2 \rfloor / n) & \text{if } j \text{ is odd} . \end{cases}$$

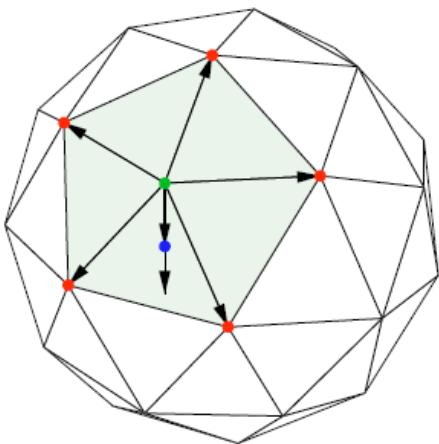


$$\Delta x = -K x$$

$$K = \begin{pmatrix} 2 & -1 & & & -1 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ -1 & & & -1 & 2 \end{pmatrix}$$

Spectral methods on graphs

- The graph Laplacian
 - Multiple definitions have been proposed
 - The straight-forward generalization is known as the combinatorial Laplacian
 - Also called unnormalized Laplacian



$$\Delta = \mathbf{D} - \mathbf{A}$$

$$A_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are neighbors} \\ 0 & \text{otherwise} \end{cases}$$

$$D_{ij} = \begin{cases} d_i = |N(i)| & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

1-D case

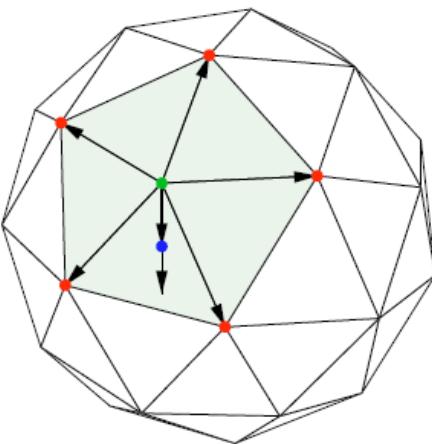
$$\Delta x = -K x$$

$$K = \begin{pmatrix} 2 & -1 & & & & -1 \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 \\ -1 & & & -1 & -1 & 2 \end{pmatrix}$$



Spectral methods on graphs

- The graph Laplacian
 - Multiple definitions have been proposed
 - The straight-forward generalization is known as the combinatorial Laplacian
 - Also called unnormalized Laplacian
 - The adjacency can be further generalized to edge weights \mathbf{W}



$$\Delta = \mathbf{D} - \mathbf{W} \quad \leftarrow$$

$$W_{ij} = w(e_{ij}) = w_{ij}$$

$w : E \rightarrow \mathbb{R}^+$, whenever $(i, j) \in E$.

$$D_{ii} = \sum_{j \in N(i)} w_{ij}.$$

Spectral methods on graphs

- The graph Laplacian
 - Multiple definitions have been proposed
 - The straight-forward generalization is known as the combinatorial Laplacian

$$\Delta = D - A$$

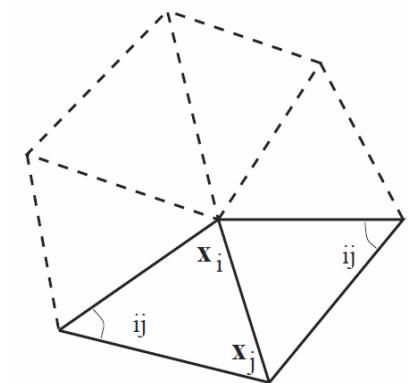
- Normalized Laplacian

$$\Delta = I - D^{-1/2}AD^{-1/2}$$

- Laplace-Beltrami Operator

- Geometric Laplacian

$$L(\mathbf{x}_i) = \frac{1}{2A_M} \sum_{j \in N_1(\mathbf{x}_i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{x}_i - \mathbf{x}_j)$$





Spectral methods on graphs

- The Laplacian on a graph of n vertices
 - Can be decomposed in n eigenvectors

$$\Delta \phi_k = \lambda_k \phi_k, \quad k = 1, 2, \dots$$

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

- Eigenvectors are real and orthonormal

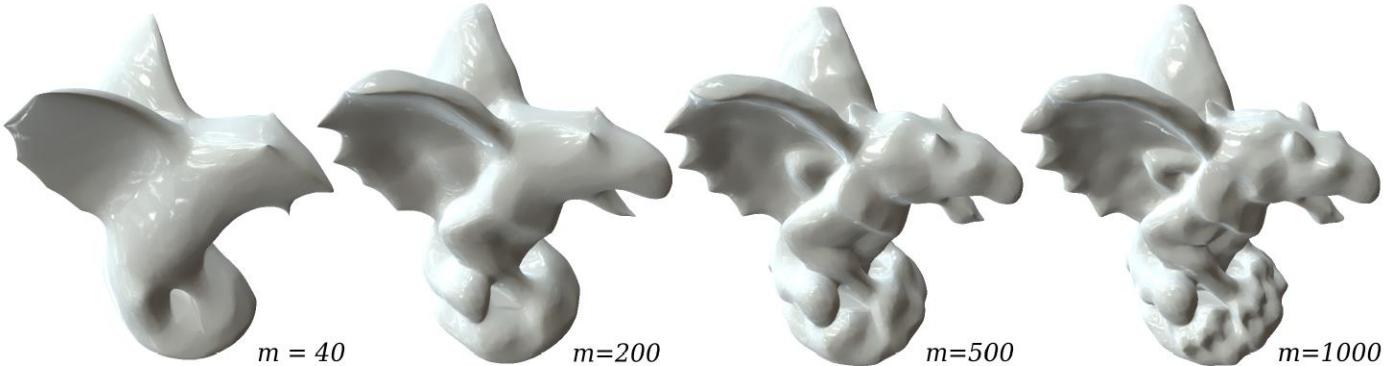
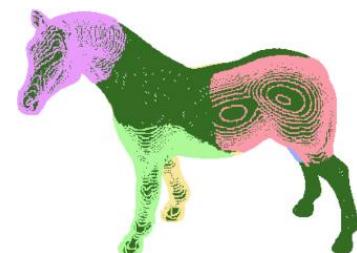
$$\langle \phi_k, \phi_{k'} \rangle_{L^2(\mathcal{V})} = \delta_{kk'}$$

- The eigenvectors of the Laplacian are analogous to the Fourier basis functions on the graph
 - Later, we take advantage of this and use the **convolution theorem**



Spectral methods on graphs

- Have been applied to a wide variety of problems
 - Mesh segmentation & correspondence
 - Surface smoothing and reconstruction
 - Watermarking



Reconstructions obtained with an increasing number of eigenfunctions.



Spectral convolution

- Given two continuous functions f, g
 - Their convolution defined as

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

- Maps into a product in the Fourier domain

$$\mathcal{F}\{f \star g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

- In the case of discrete signals (vectors)

$$\mathbf{f} = (f_1, \dots, f_n)^\top \text{ and } \mathbf{g} = (g_1, \dots, g_n)^\top$$

$$\mathbf{f} \star \mathbf{g} = \Phi(\Phi^\top \mathbf{g} \circ \Phi^\top \mathbf{f})$$

Fourier basis

Spectral convolution

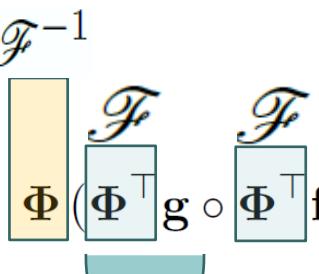
- In graphs, we work *by analogy* to the discrete case

$$\mathbf{f} = (f_1, \dots, f_n)^\top \text{ and } \mathbf{g} = (g_1, \dots, g_n)^\top$$

$$\mathbf{f} \star \mathbf{g} = \Phi(\Phi^\top \mathbf{g} \circ \Phi^\top \mathbf{f})$$

- Where the Fourier basis comes from the Laplacian

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

$$\mathcal{F}^{-1}$$


- Therefore $\mathbf{f} \star \mathbf{g} = \Phi(\Phi^\top \mathbf{g} \circ \Phi^\top \mathbf{f})$

$$\hat{g}_\theta(\Lambda) = \text{diag}(\theta)$$

$$= \Phi \hat{g}(\Lambda) \Phi^\top \mathbf{f}$$

$\theta \in \mathbb{R}^n$ is a vector of Fourier coefficients

Spectral convolution

- In graphs, we work *by analogy* to the discrete case

$$\mathbf{f} = (f_1, \dots, f_n)^\top \text{ and } \mathbf{g} = (g_1, \dots, g_n)^\top$$

$$\mathbf{f} \star \mathbf{g} = \Phi(\Phi^\top \mathbf{g} \circ \Phi^\top \mathbf{f})$$

- Where the Fourier basis comes from the Laplacian

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

- Theretore

$$\mathbf{f} \star \mathbf{g} = \Phi(\Phi^\top \mathbf{g} \circ \Phi^\top \mathbf{f})$$

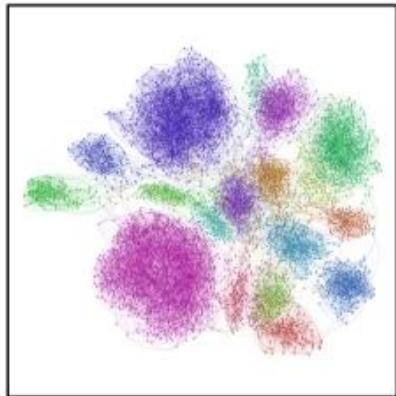
$$\hat{g}_\theta(\Lambda) = \text{diag}(\theta)$$

$$= \Phi \hat{g}(\Lambda) \Phi^\top \mathbf{f} = \hat{g}(\Phi \Lambda \Phi^\top) \mathbf{f} = \hat{g}(\Delta) \mathbf{f}$$

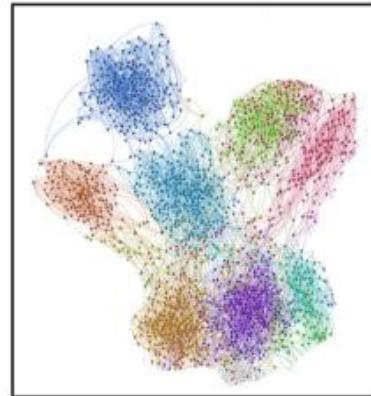
$\theta \in \mathbb{R}^n$ is a vector of Fourier coefficients

● ● ● | Graph pooling

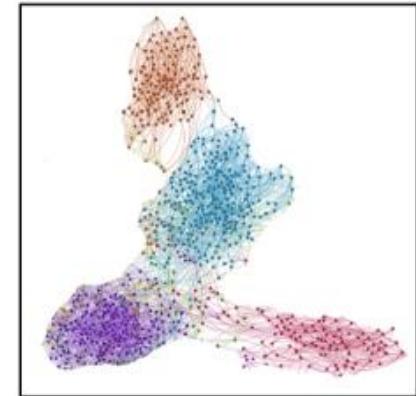
- Equivalent to graph downsampling
 - Graph coarsening
 - Graph partitioning
 - Lots of research
 - But NP-hard problem



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

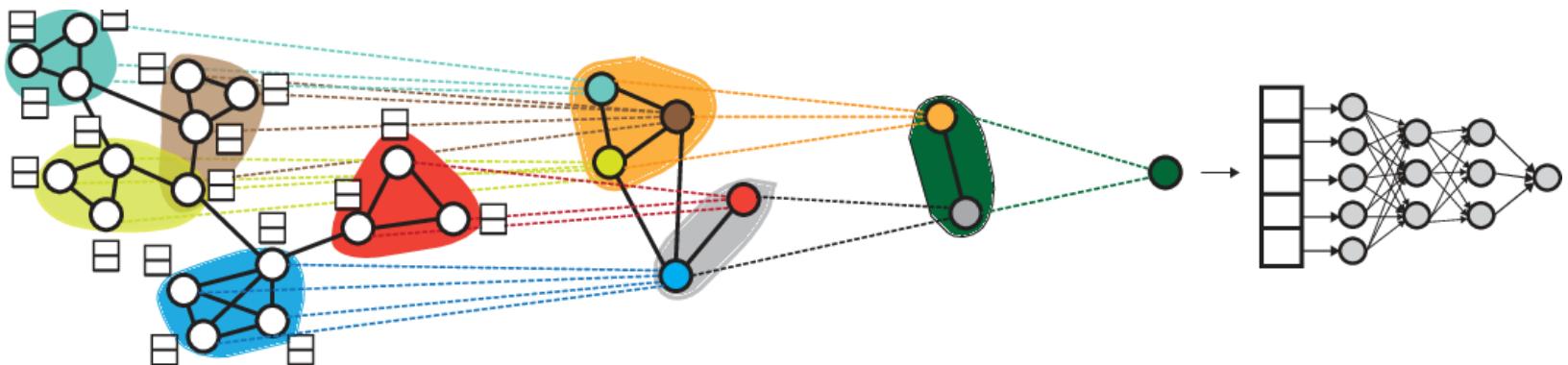


$$G^{l=1}$$



$$G^{l=2}$$

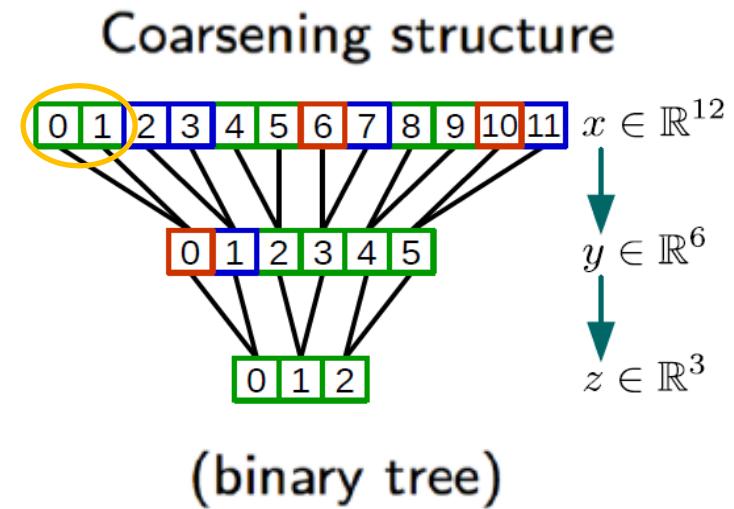
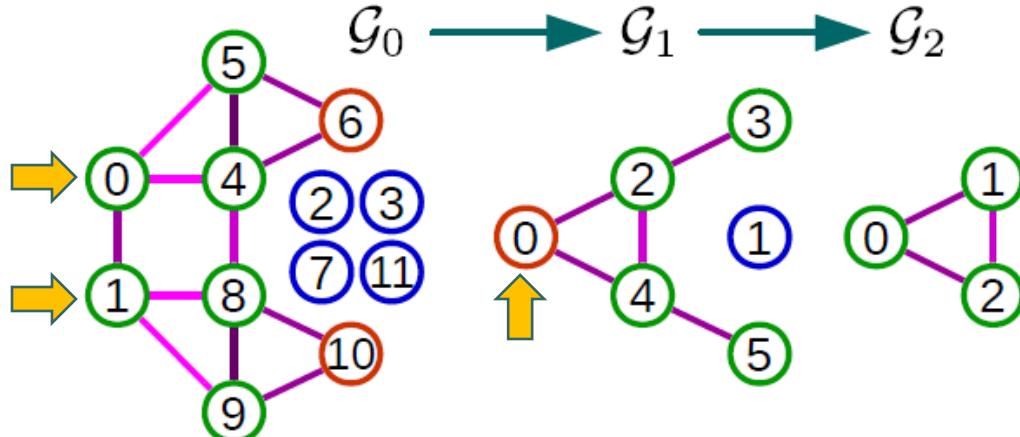
Graph pooling



- Approximate solutions
 - Not guaranteed to be optimal
 - But often suffice for our needs
- Structured pooling
 - Arrangement of the node indexing
 - Adjacent nodes are hierarchically merged

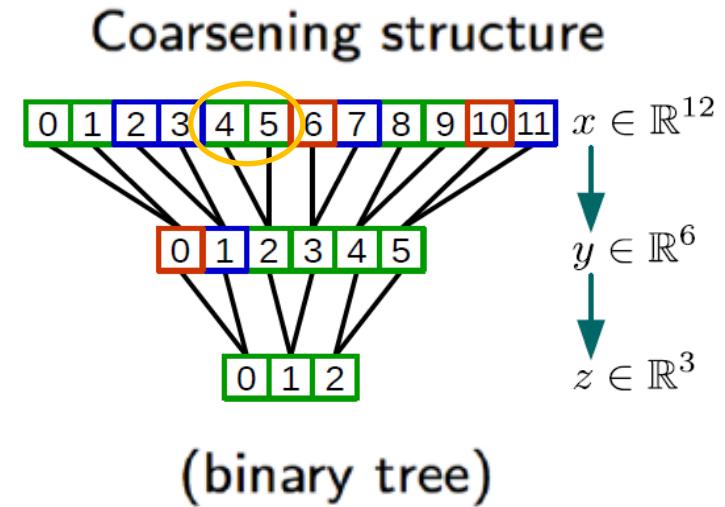
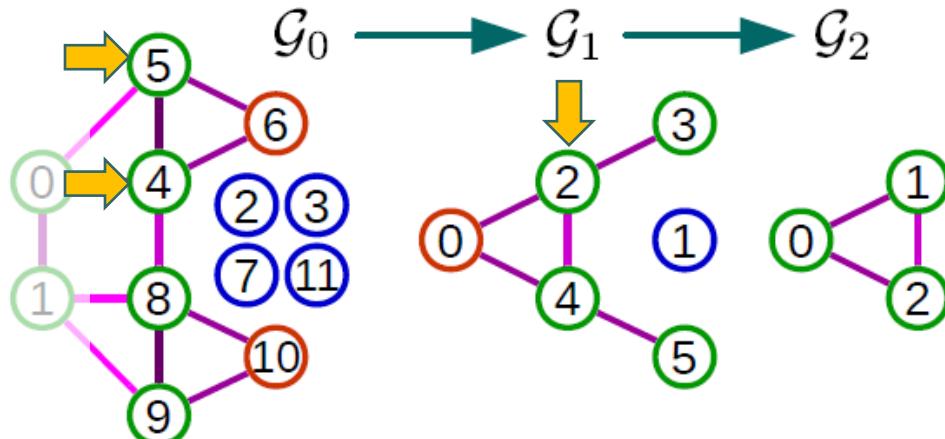
Graph pooling

- Structured pooling
 - Arrangement of the node indexing
 - Adjacent nodes are hierarchically merged
 - Requires adding “ghost” nodes
 - As efficient as a 1D-Euclidean grid pooling



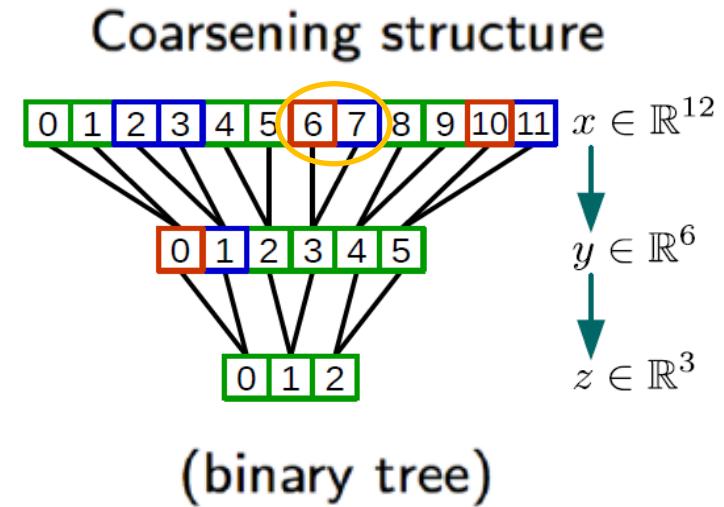
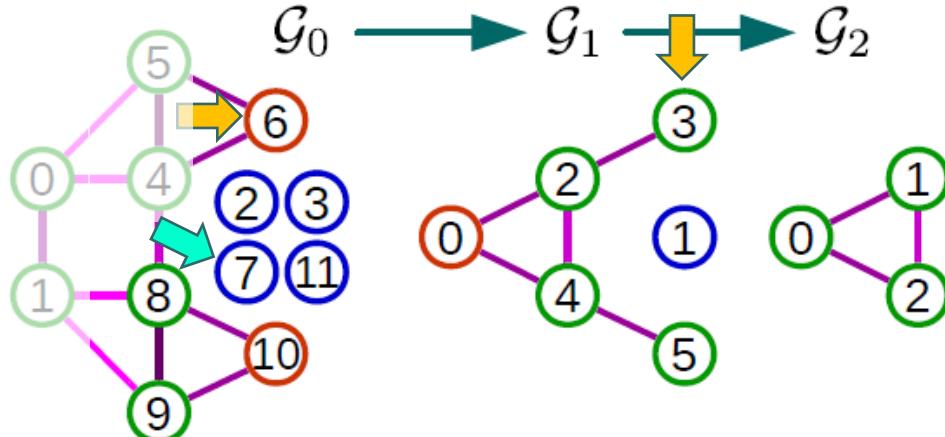
Graph pooling

- Structured pooling
 - Arrangement of the node indexing
 - Adjacent nodes are hierarchically merged
 - Requires adding “ghost” nodes
 - As efficient as a 1D-Euclidean grid pooling



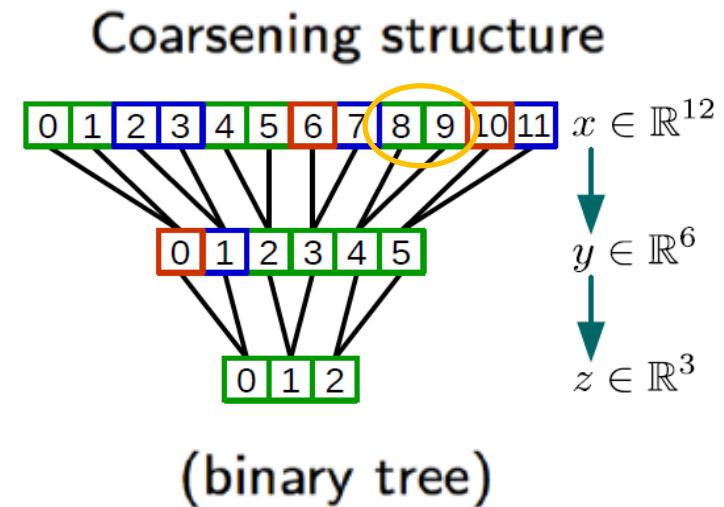
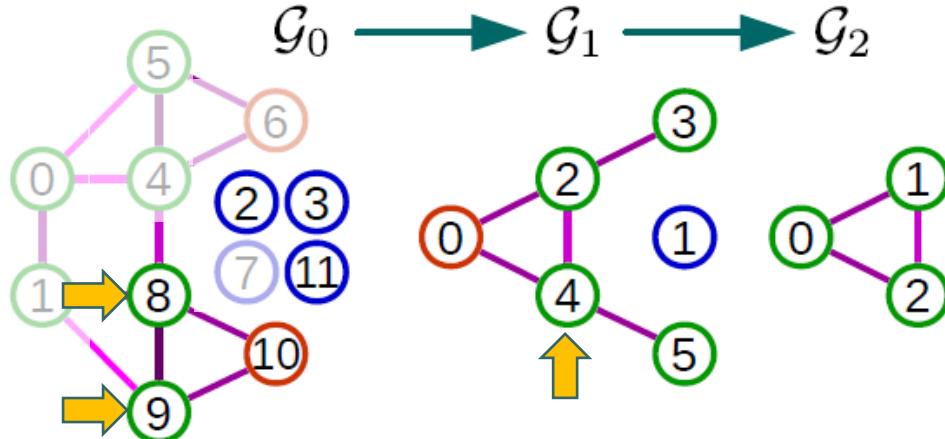
Graph pooling

- Structured pooling
 - Arrangement of the node indexing
 - Adjacent nodes are hierarchically merged
 - Requires adding “ghost” nodes
 - As efficient as a 1D-Euclidean grid pooling



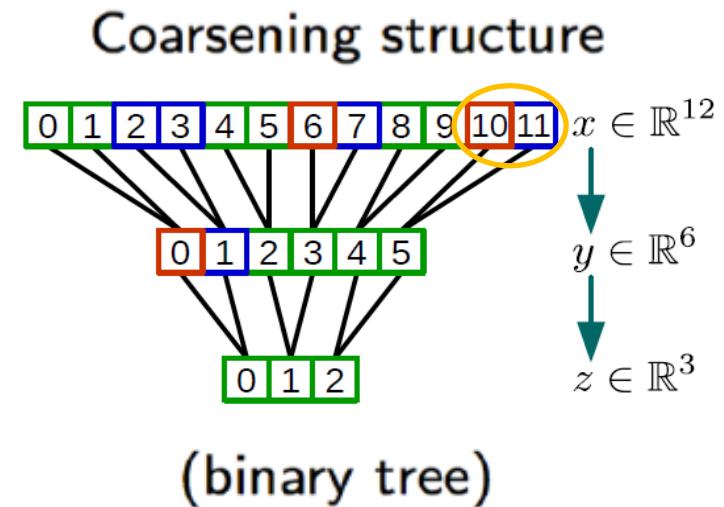
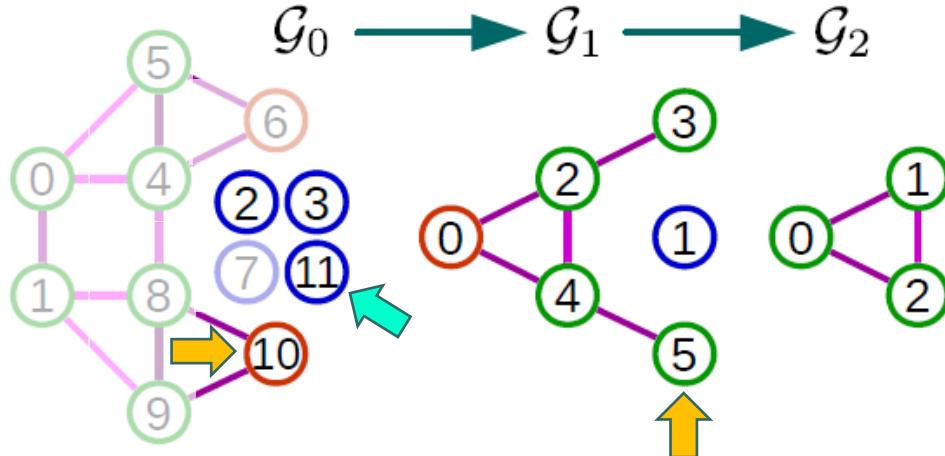
Graph pooling

- Structured pooling
 - Arrangement of the node indexing
 - Adjacent nodes are hierarchically merged
 - Requires adding “ghost” nodes
 - As efficient as a 1D-Euclidean grid pooling



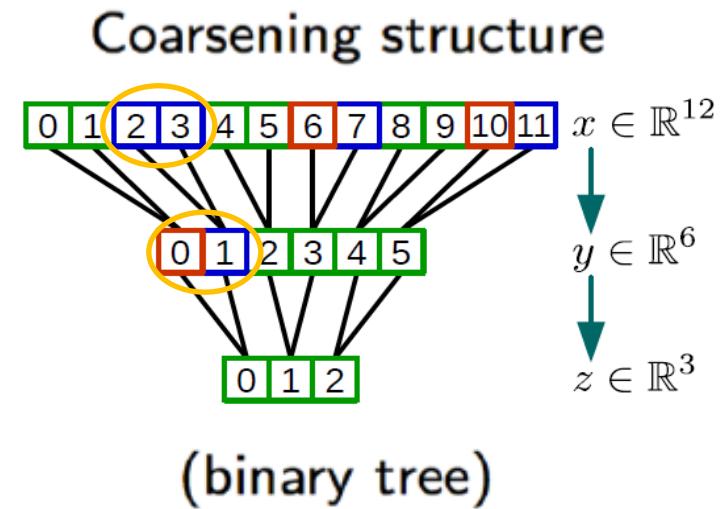
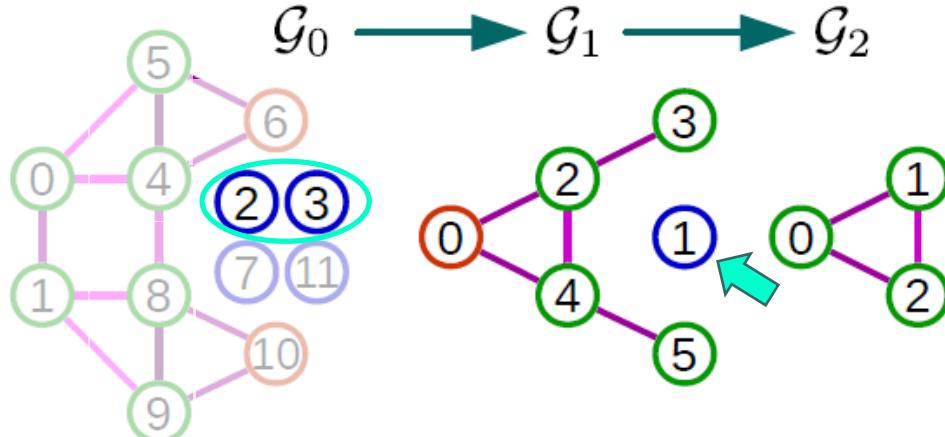
Graph pooling

- Structured pooling
 - Arrangement of the node indexing
 - Adjacent nodes are hierarchically merged
 - Requires adding “ghost” nodes
 - As efficient as a 1D-Euclidean grid pooling



Graph pooling

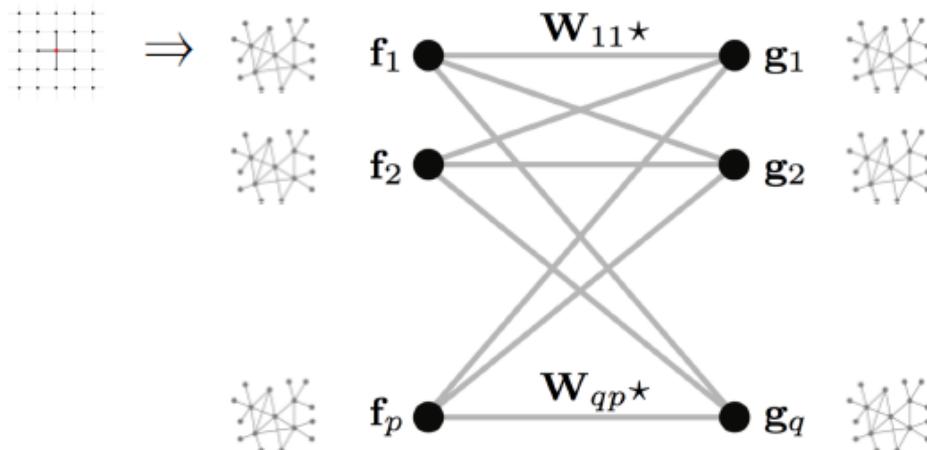
- Structured pooling
 - Arrangement of the node indexing
 - Adjacent nodes are hierarchically merged
 - Requires adding “ghost” nodes
 - As efficient as a 1D-Euclidean grid pooling



Vanilla graph CNNs

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

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



Vanilla spectral graph CNNs

- The convolutional layer in the spatial domain

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

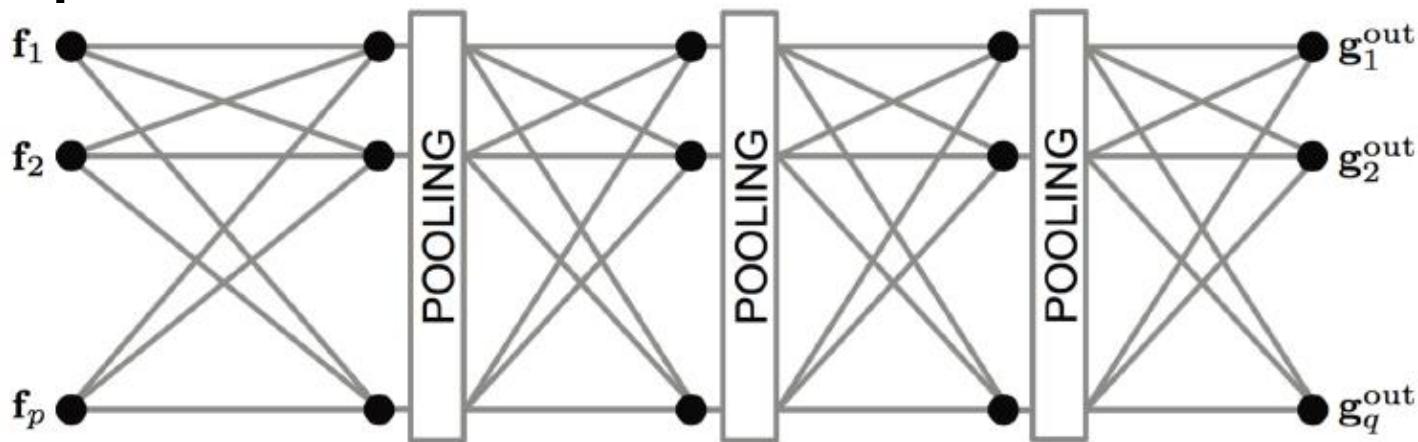
- Can be replaced by a filter expressed in the spectral domain

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

$$\Phi \hat{\mathbf{W}}_{l,l'} \Phi^\top \mathbf{f}_{l'}$$

- ($n \times n$) diagonal matrix
- Filter coefficients in the spectral domain

Graph Compositional layers



f_l = l l -th data feature on graphs, $\dim(f_l) = n \times 1$

$g_l^{(k)}$ = l l -th feature map, $\dim(g_l^{(k)}) = n \times 1$

Convolutional layer
$$g_l^{(k)} = \xi \left(\sum_{l'=1}^{q^{(k-1)}} \Phi \hat{W}_{l,l'}^{(k)} \Phi^\top g_{l'}^{(k-1)} \right) f_l$$

Activation, e.g.

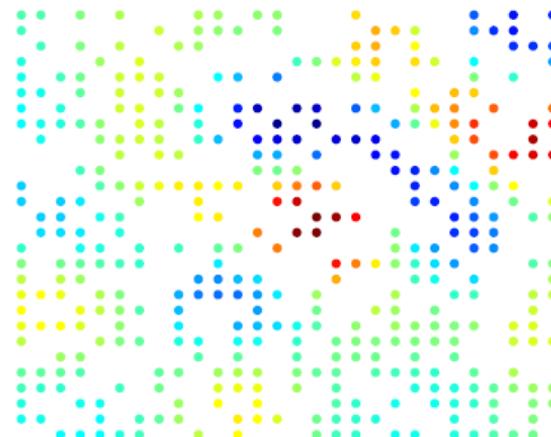
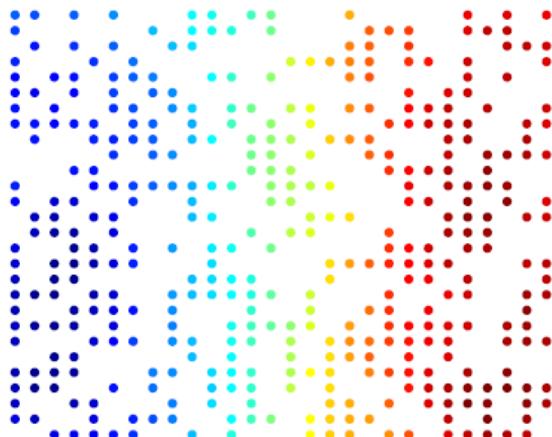
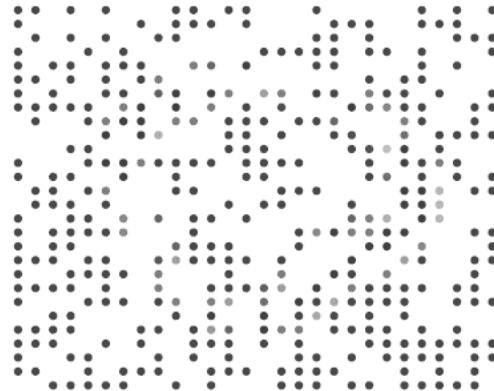
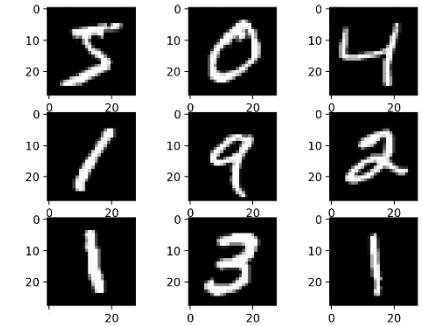
$$\xi(x) = \max\{x, 0\} \quad \text{rectified linear unit (ReLU)}$$

Pooling

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

Graph CNNs on a synthesized graph

- MNIST digits database
 - Images were sub-sampled
 - 400 points
 - Non-regular graph
 - Example of the first two eigenvectors of the Laplacian

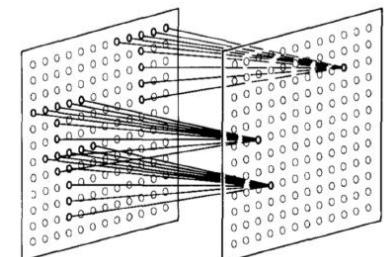


Spectral domain issues

- Direct operation in spectral domain

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

- Allows to compute convolutions on a graph
- But is computationally expensive
 - $O(n)$ parameters to be learned in each layer
 - $O(n^2)$ complexity to transform between domains
 - Fourier and inverse Fourier
 - $O(n^3)$ complexity for the eigen-decomposition of the Laplacian
- No guarantee of spatial localization of filters



Toward spatial localization

- Localization in space
 - Implies smoothness in the spectrum
- SplineNet
 - Re-define the spectral filter parameters
 - As linear combinations of smooth kernel functions
 - Based on splines
- ChebNet
 - Re-define the spectral filter parameters
 - With polynomials of the Laplacian eigenvalues

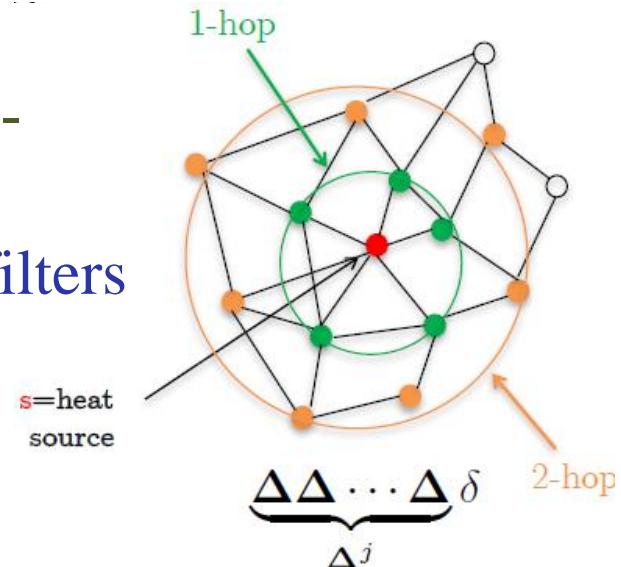
$$w_{\alpha}(\lambda) = \sum_{j=0}^r \alpha_j \lambda^j$$

ChebNet

- Re-define the spectral filter parameters
 - With polynomials of the Laplacian eigenvalues

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

- Spectral filters of this type will be r -localized (in space)
 - Equivalent spatial coefficients beyond r -neighbors will be zero
- Exact control of the spatial support of the filters
- Parameter complexity reduced to $O(r)$



ChebNet

- Chebynet

- Spectral GCNN

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

- Spectral polynomials

$$\mathbf{g}_l^{(k)} = \xi \left(\sum_{l'=1}^{q^{(k-1)}} w_{\alpha}^{(k)}(\Delta) \mathbf{g}_{l'}^{(k-1)} \right)$$

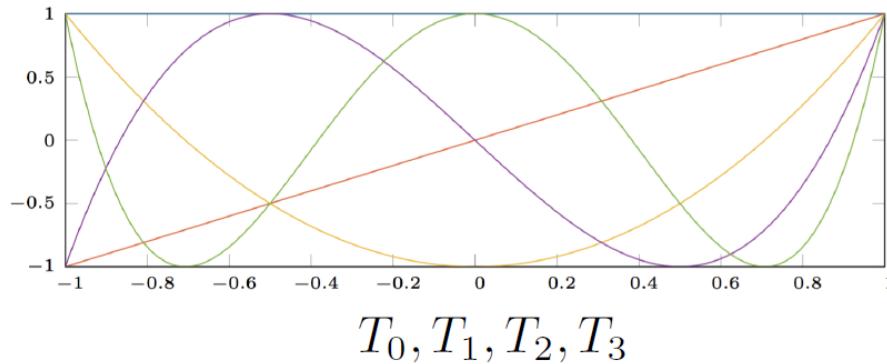
- Therefore

- ~~O(n) parameters $\Rightarrow O(r)$ parameters~~
 - ~~O(n^2) complexity to transform between domains~~
 - Reduced to approximately O(n) for sparse graphs
 - ~~O(n^3) complexity for eigen-decomposition of Laplacian~~
 - No need to explicitly compute the spectrum of the graph

ChebNet

- Recursive formulation of filters
 - Using Chebyshev polynomial expansion
 - More stable under perturbations
 - Better behavior for optimization
 - Approximate solution
 - The Laplacian must be scaled appropriately

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



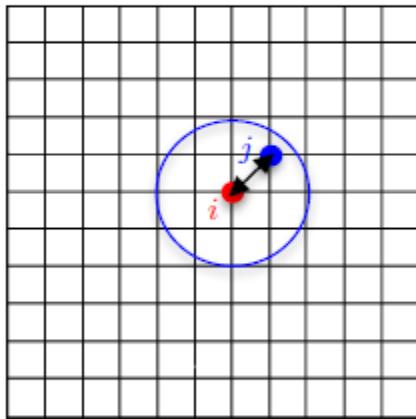
$$T_j(\tilde{\lambda}) = 2\tilde{\lambda}T_{j-1}(\tilde{\lambda}) - T_{j-2}(\tilde{\lambda})$$

$$T_0(\tilde{\lambda}) = 1, \quad T_1(\tilde{\lambda}) = \tilde{\lambda}$$

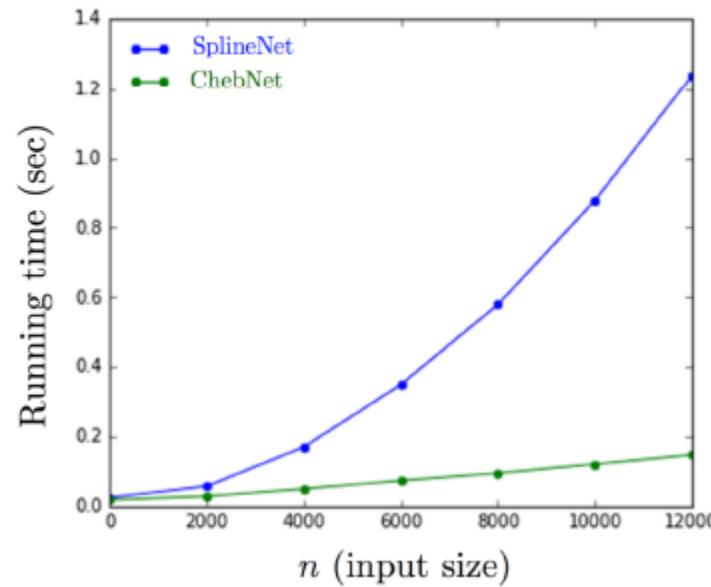
Experiments in a Euclidean grid

- Digit recognition
 - MNIST database

Graph: a 8-NN
graph of the
Euclidean grid

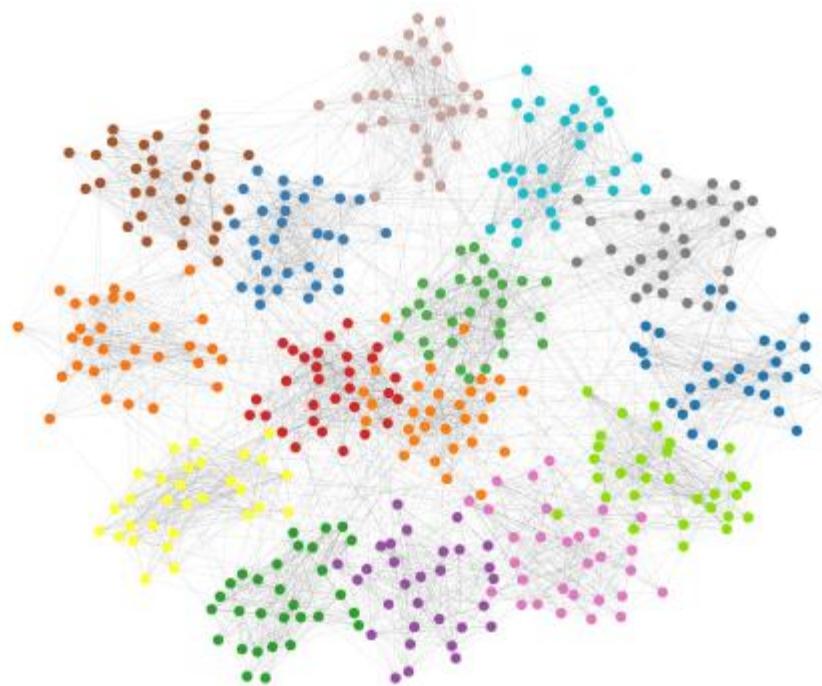


- Running time



••• Beyond ChebNet I: CayleyNet

- Some graphs are difficult for ChebNet
 - For example, community graphs
 - In such cases ChebNet requires relatively high-order polynomials



Synthetic graph with 15 communities

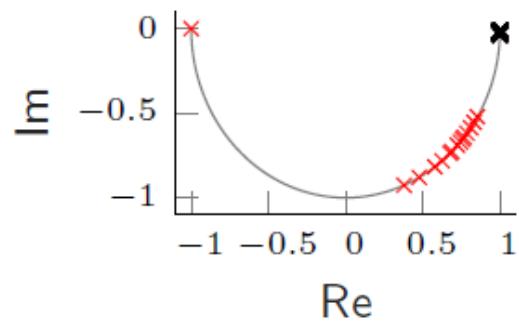
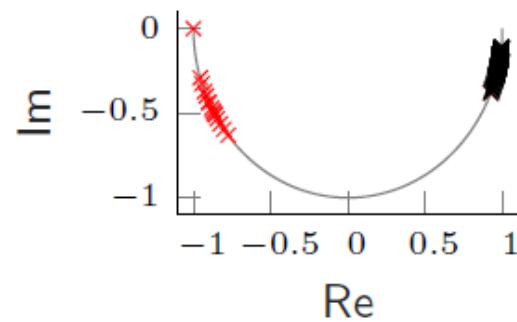
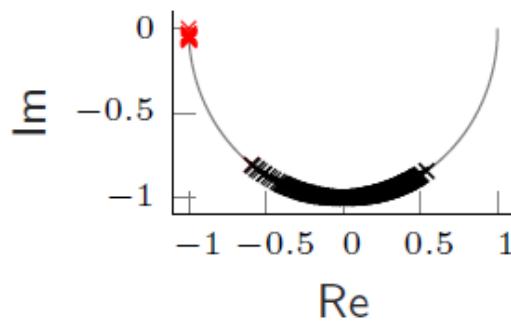
• • • Beyond ChebNet I: CayleyNet

- CayleyNet
 - Use the Cayley transform

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

- To map the (scaled) eigenvalues of the Laplacian
 - Non-linearly to the unit circle
 - Spectral zoom

$$C(h\lambda) = (h\lambda - i)(h\lambda + i)^{-1}$$



Cayley transform $C(h\lambda)$ for (left-to-right) $h = 0.1, 1$, and 10
of the 15-communities graph Laplacian spectrum



Beyond ChebNet I: CayleyNet

- Cayley polynomials of order r
 - Are a family of real-valued rational functions
 - With complex coefficients

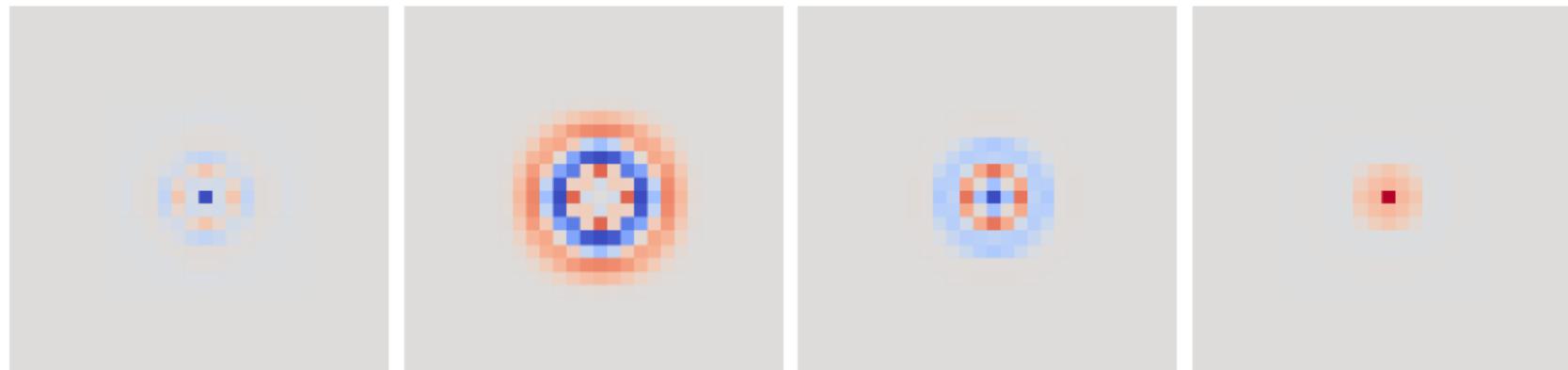
$$\begin{aligned}\tau_{\mathbf{c}}(h\Delta) &= \Phi \left(c_0 + \sum_{j=1}^r c_j C(h\Lambda)^j + \sum_{j=1}^r \bar{c}_j C(h\Lambda)^{-j} \right) \Phi^T = \\ &= c_0 + \sum_{j=1}^r c_j C(h\Delta)^j + \sum_{j=1}^r \bar{c}_j C(h\Delta)^{-j}.\end{aligned}$$

ChebNet vs CayleyNet on a Euclidean grid

- Chebyshev filters of order 3

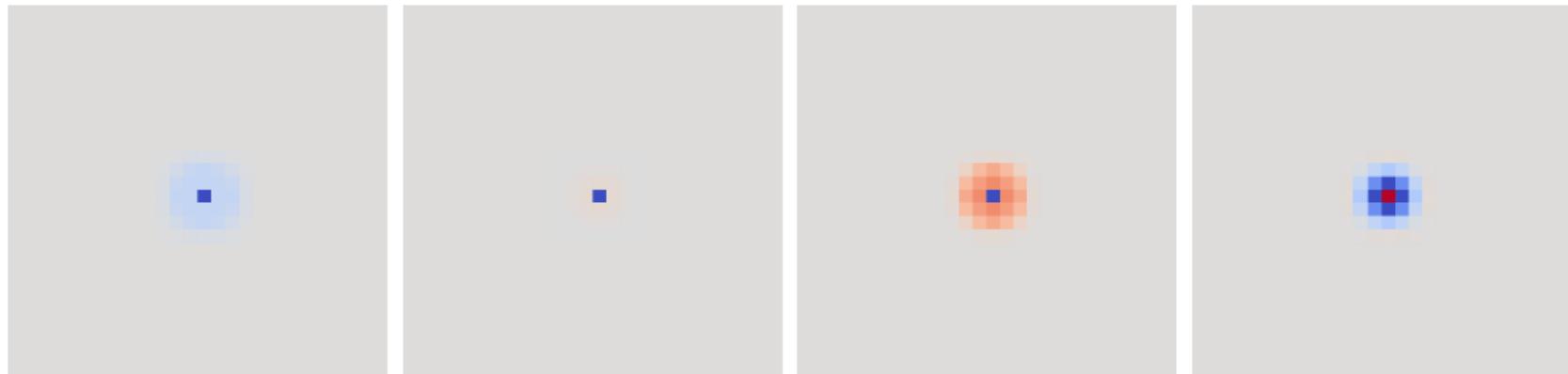


- Chebyshev filters of order 7

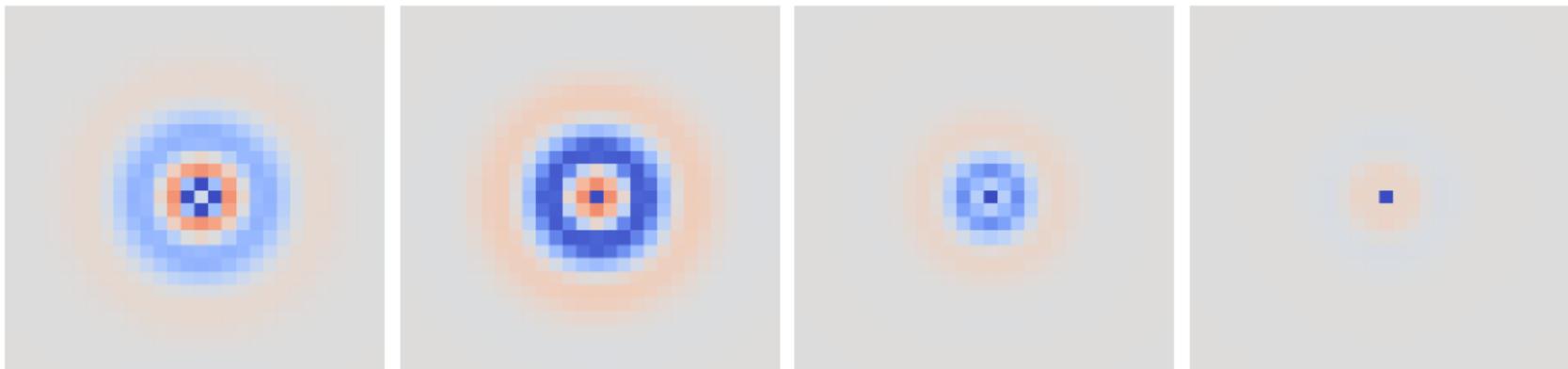


ChebNet vs CayleyNet on a Euclidean grid

- Chebyshev filters of order 3

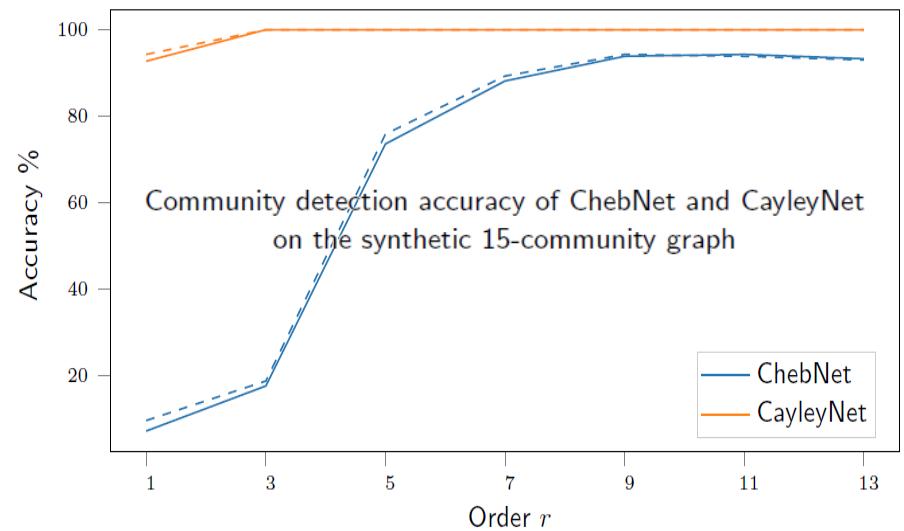


- Cayley filters of order 3



ChebNet vs CayleyNet

- Cayley polynomials
 - Are richer than Chebychev polynomials
 - Can be understood as a generalization
 - Improved performance
- Computational cost
 - ChebNet is faster
 - Laplacian inversion
 - Exact $O(n^3)$
 - Approximate solutions $O(n)$



Limitations of Spectral Graph CNNs

- Poor generalization to different graphs
 - Based on the spectrum of the Laplacian
 - Have been developed for fixed graphs only
 - Fourier basis not stable under graph perturbation
 - Difficult to generalize to variable graphs

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

• Vertices $\mathcal{V} = \{1, \dots, n\}$

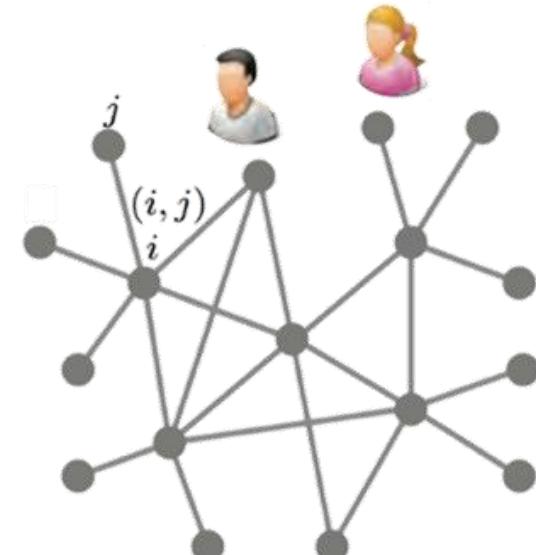
• Edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$

• Vertex weights $b_i > 0$ for $i \in \mathcal{V}$

• Edge weights $a_{ij} \geq 0$ for $(i, j) \in \mathcal{E}$

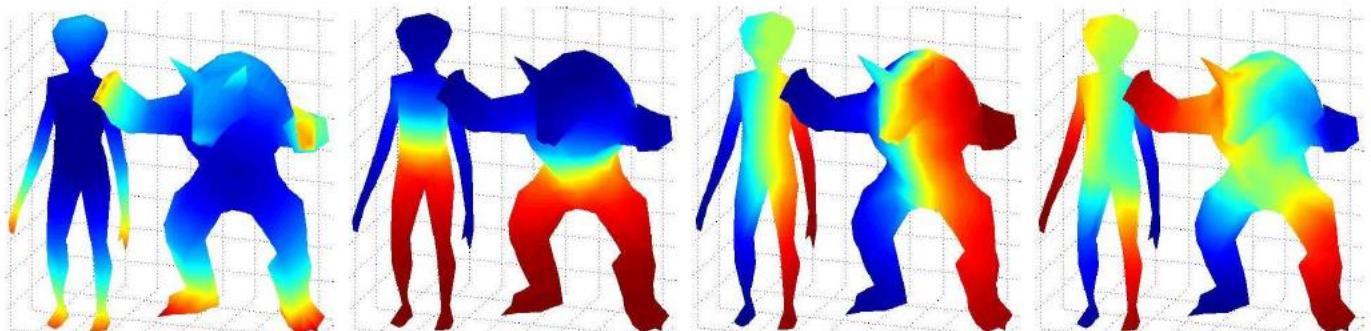
• Vertex fields $L^2(\mathcal{V}) = \{f : \mathcal{V} \rightarrow \mathbb{R}^h\}$

Represented as $\mathbf{f} = (f_1, \dots, f_n)$



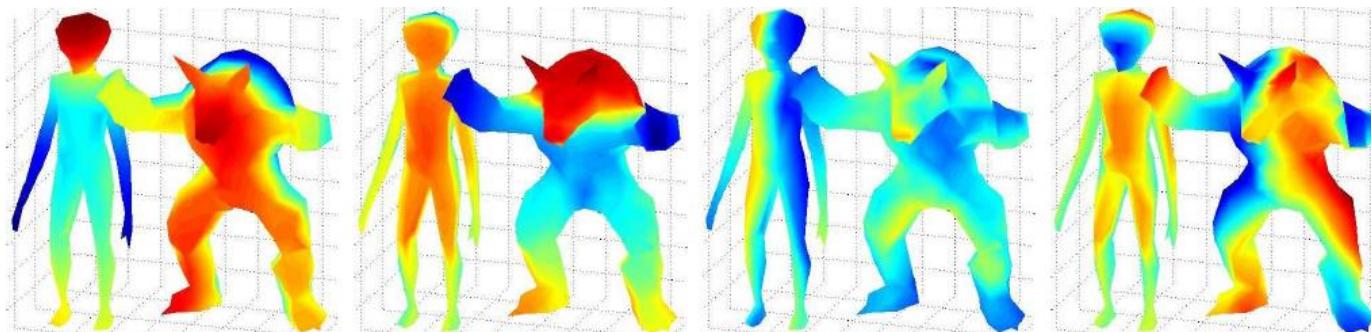
Laplace-Beltrami Spectrum

- Takes into account the geometry of the graph
 - Not only the connectivity
 - Different geometries will have different spectrum
 - But the spectrum should be equivalent
 - Example of the first Laplace Beltrami eigenvectors
 - For two different shapes
 - Most of the times we are not so lucky.....



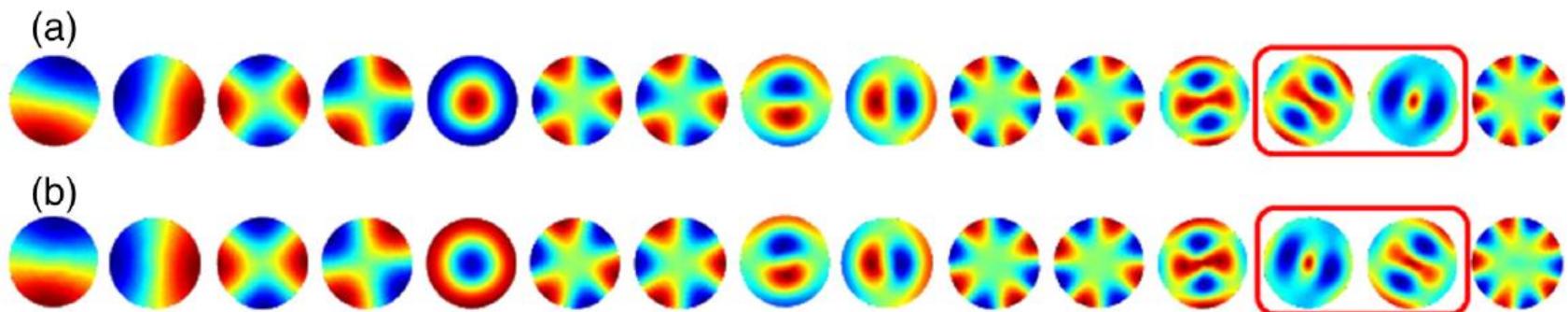
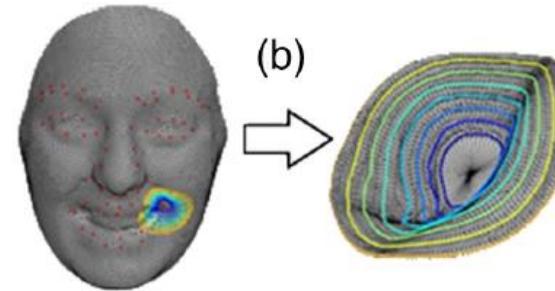
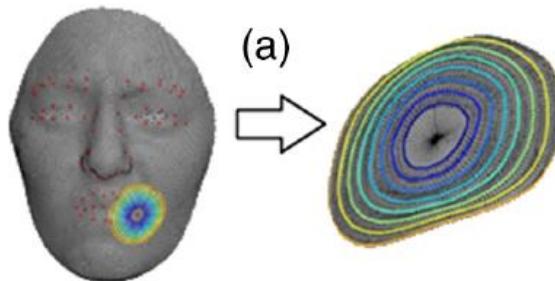
Laplace-Beltrami Spectrum

- Takes into account the geometry of the graph
 - Not only the connectivity
 - Different geometries will have different spectrum
 - But the spectrum should be equivalent
 - Example of the first Laplace Beltrami eigenvectors
 - For two different shapes
 - Eigenvectors 5 to 8



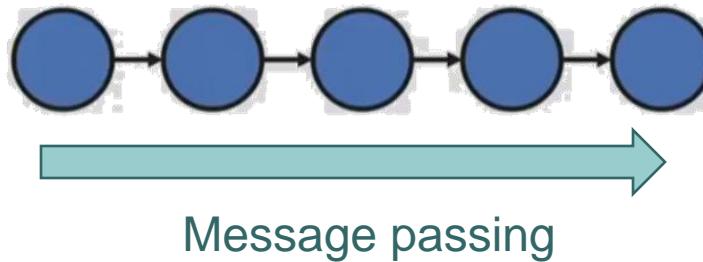
Laplace-Beltrami Spectrum

- Example of the first Laplace Beltrami eigenvectors
 - For two different local patches on the facial surface

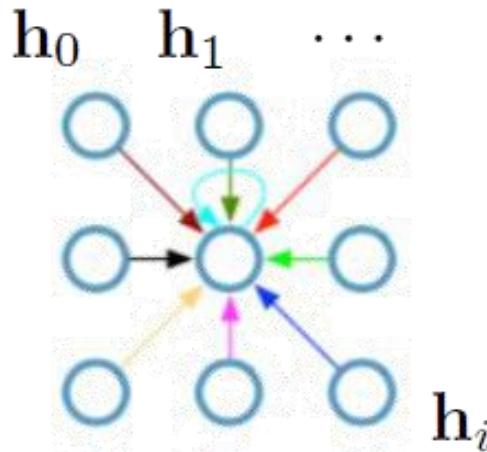


RNN vs CNN motivation for Graph Neural Networks

- A Recurrent Neural Network (RNN)
 - Can be thought of as a special type of graph



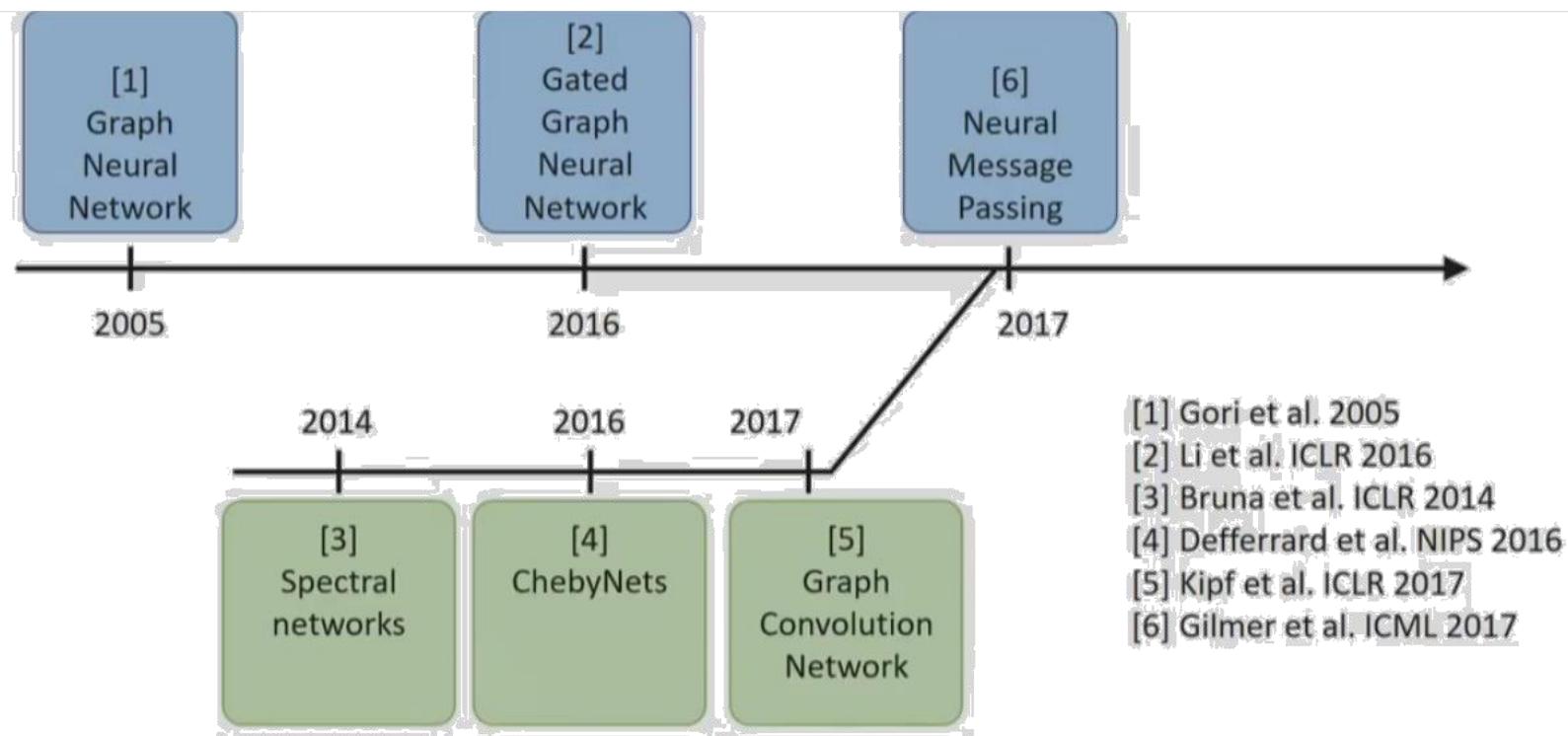
- Graph Convolutional Nets can be seen equivalently



$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right)$$

RNN vs CNN motivation for Graph Neural Networks

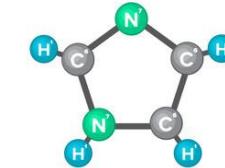
- Both approaches developed from independent paths
 - They were shown to converge to similar formulations



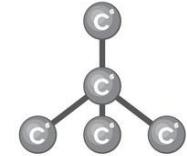


Graph Neural Networks

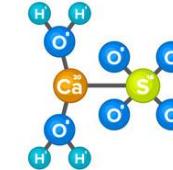
- Applications to multiple domains
 - Computer graphics
 - Body pose and gestures
 - Citation networks / Social networks
 - Matrix completion
 - Molecule structure



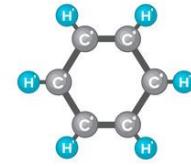
Imidazole



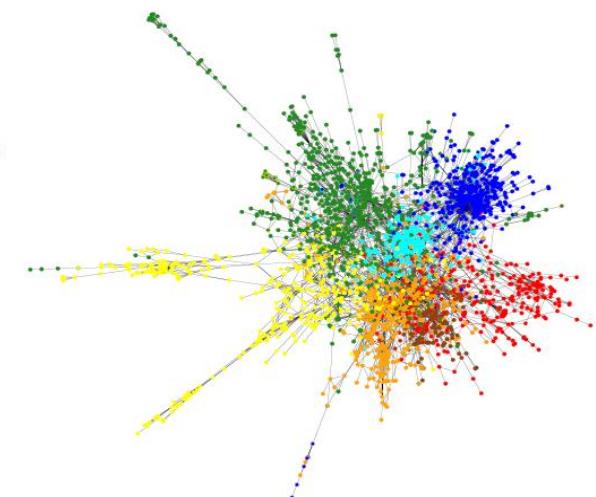
Diamond crystal



Gypsum



Benzol (Benzene)





Additional Sources

- Slides inspired from several talks available on-line
 - Xavier Bresson
 - https://www.youtube.com/watch?time_continue=2&v=v3jZRkvIOIM&feature=emb_logo
 - Federico Monti
 - <https://www.youtube.com/watch?v=nCv05re-8lQ&t=1316s>
 - Alex Gaunt
 - https://www.youtube.com/watch?time_continue=2&v=cWIeTMklzNg&feature=emb_logo



Additional Sources

- J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” in Proc. of ICLR, 2014.
- M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in Proc. of NIPS, 2016
- T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. of ICLR, 2017.
- F. Monti, M. Bronstein, and X. Bresson, “Geometric matrix completion with recurrent multi-graph neural networks,” in Proc. of NIPS, 2017,

Reviews

- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint*
- Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*