Point Cloud Processing II

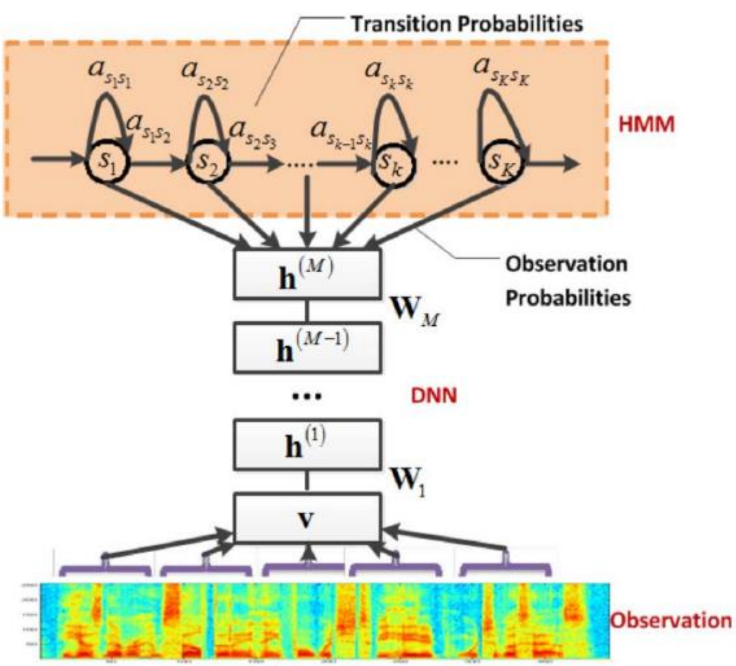
Deep Learning methods

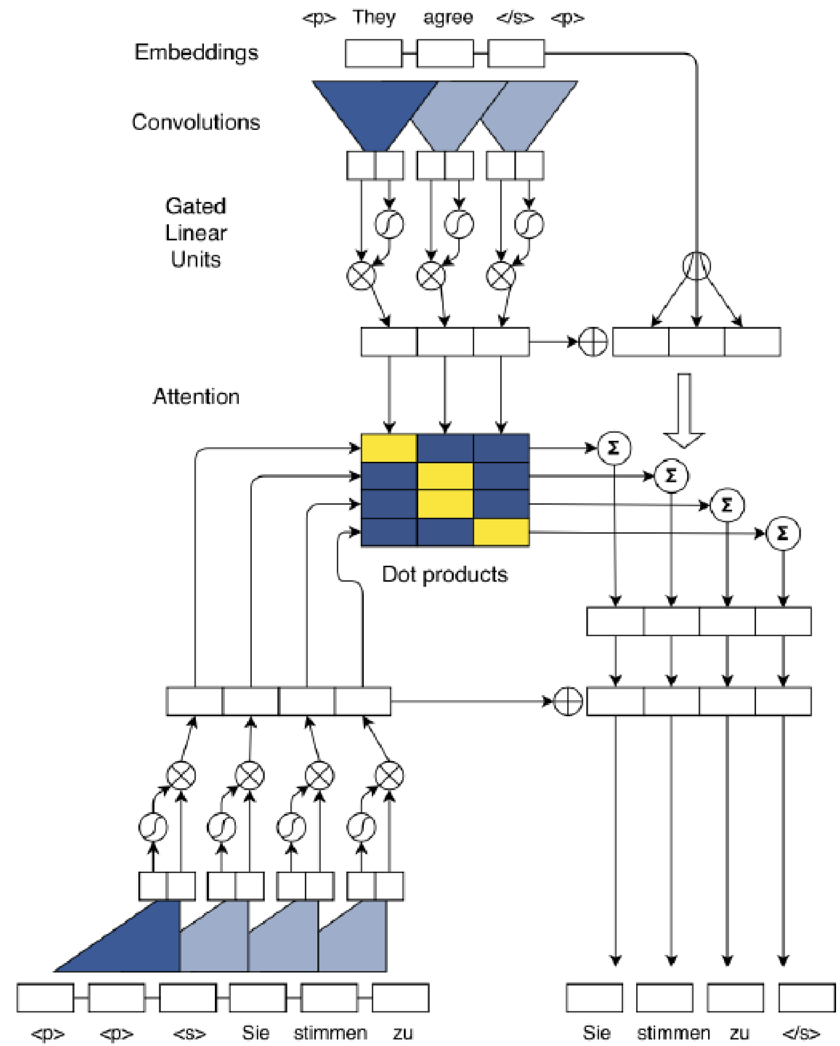
Why Deep Learning ?

- 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

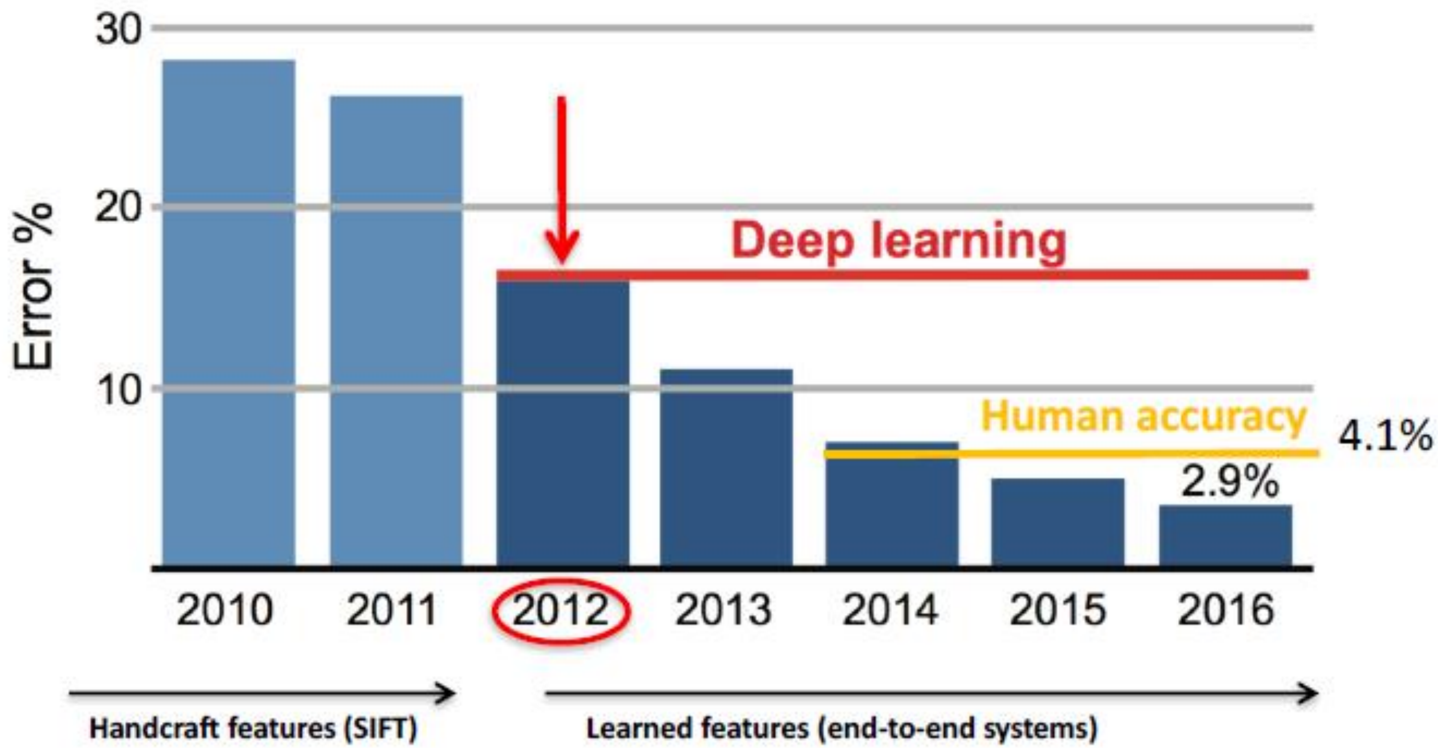
- Recurrent Neural Networks (RNNs)



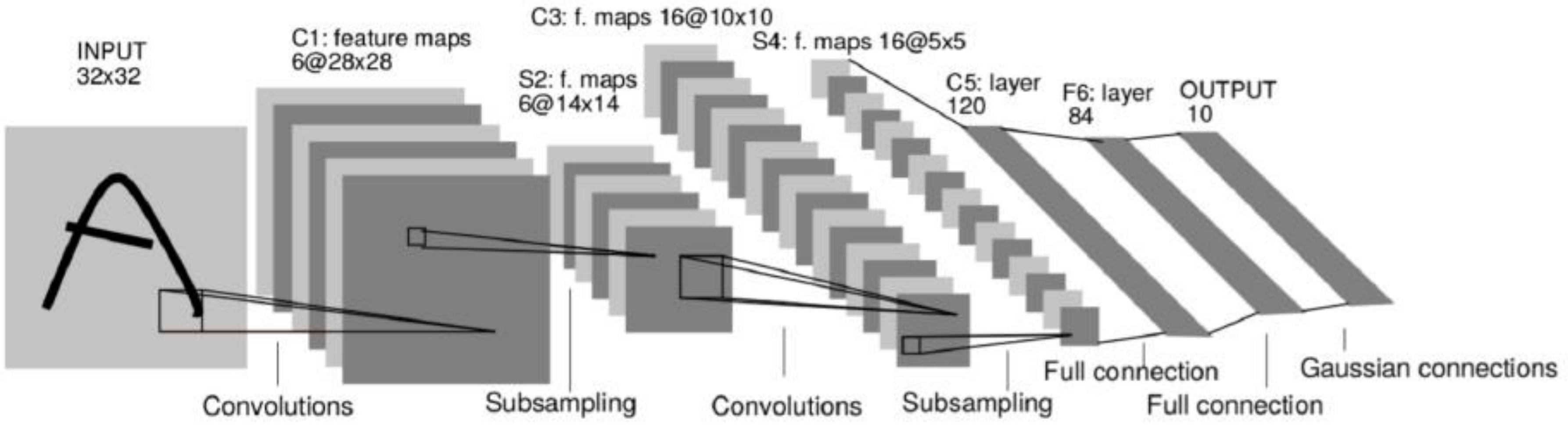


Deep learning w/images

- Convolutional Neural Networks (CNNs)



Convolutional Neural Networks (CNN)



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

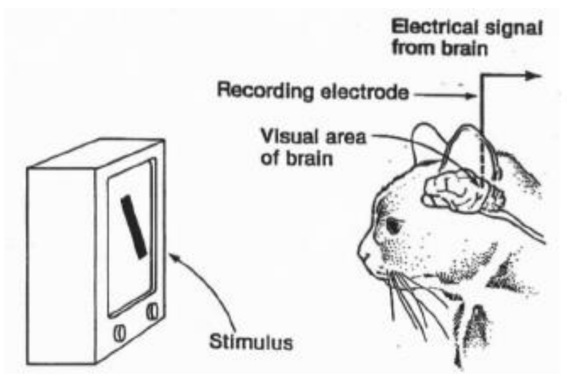
Compositional assumption

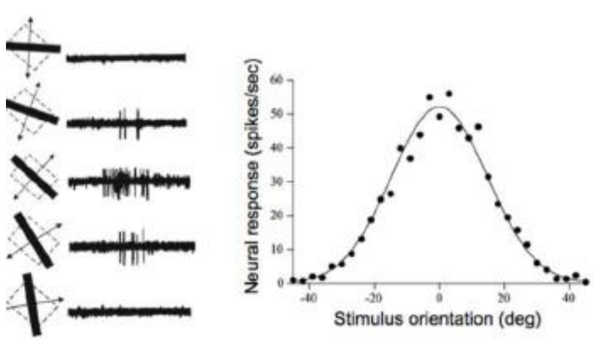
- Data (images, videos, sounds) are compositional
  - They are formed of patterns that are
    - Local
    - Stationary
    - Multi-scale (hierarchical)
- Convolutional neural networks
  - Leverage the compositionality structure of data
  - They extract compositional features

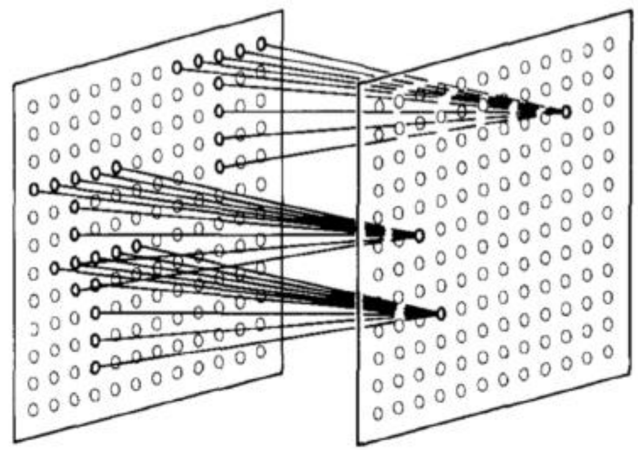
Locality

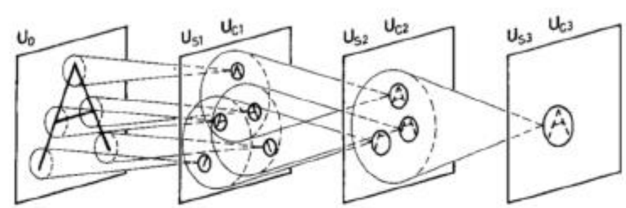
- Local
- Stationary
- Multi-scale (hierarchical)

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





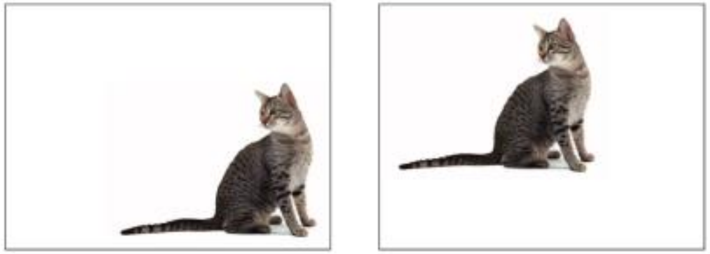




- Local
- Stationary
- Multi-scale (hierarchical)

Stationarity

- Global stationarity
  - Translation invariance

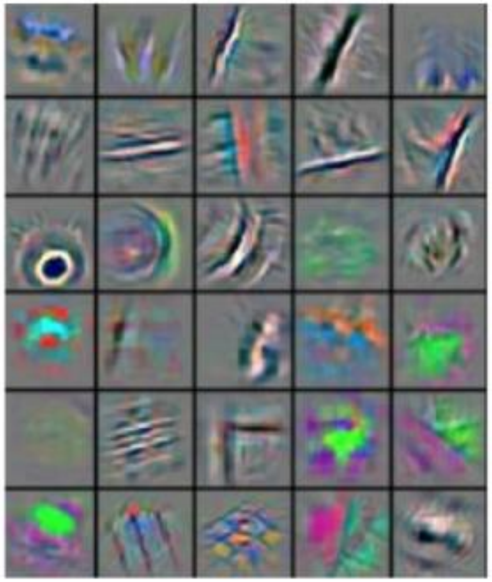
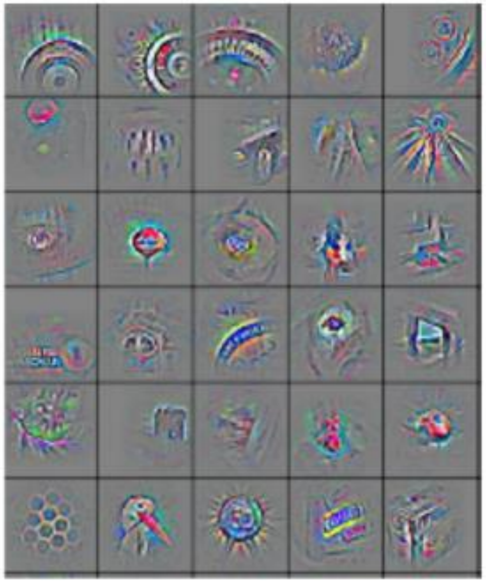


- Local stationarity
  - Similar patches are shared across the data domain
  - Local invariance
  - Good for intra-class variation

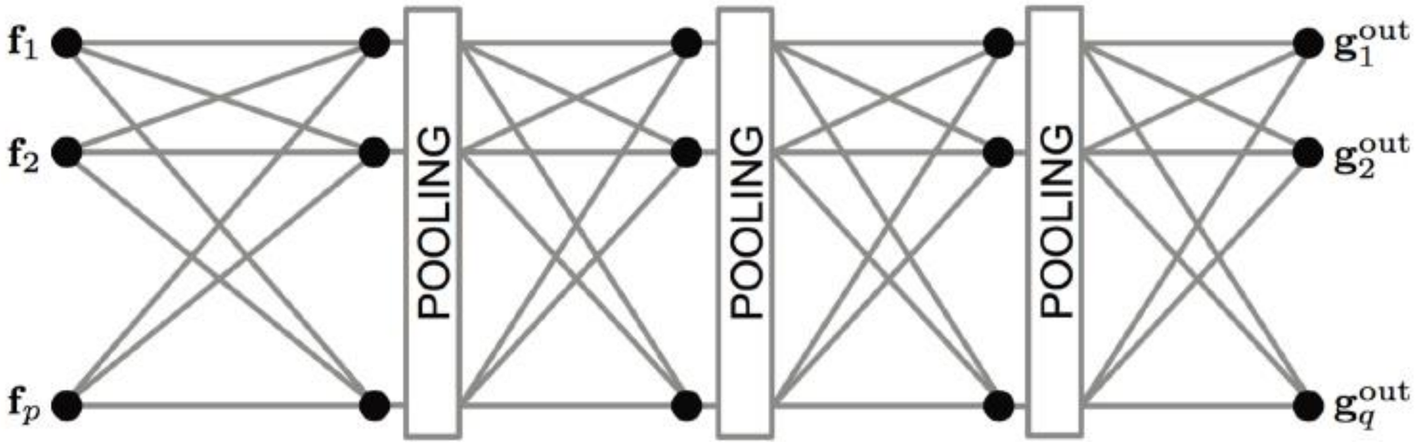
- Local
- Stationary
- Multi-scale (hierarchical)

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



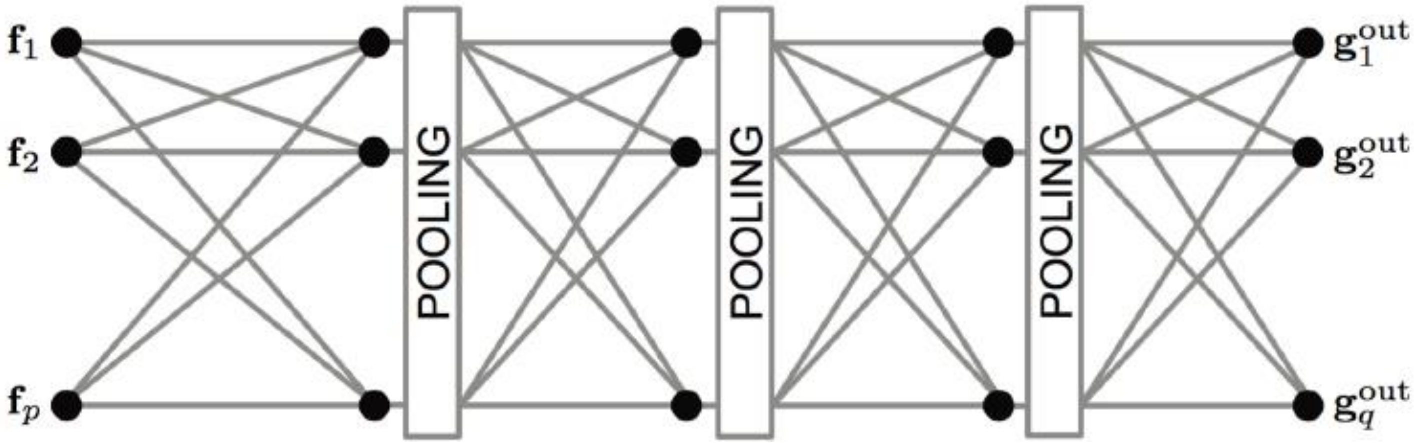
$$\begin{aligned} \mathbf{f}_l &= l\text{-th image feature (R,G,B channels), } \dim(\mathbf{f}_l) = n \times 1 \\ \mathbf{g}_l^{(k)} &= l\text{-th feature map, } \dim(\mathbf{g}_l^{(k)}) = n_l^{(k)} \times 1 \end{aligned}$$

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

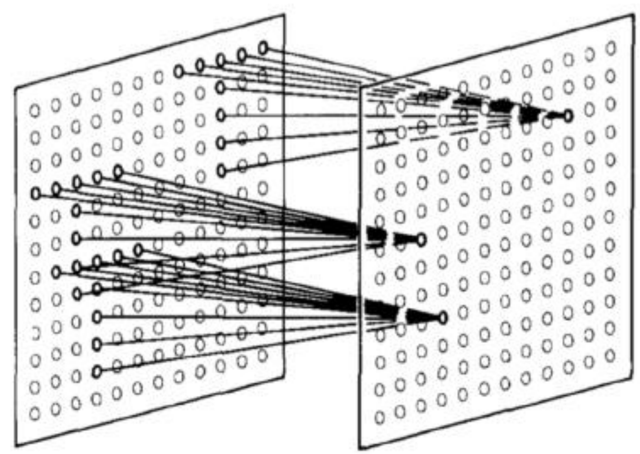
Activation, e.g. $\xi(x) = \max\{x, 0\}$ rectified linear unit (ReLU)

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

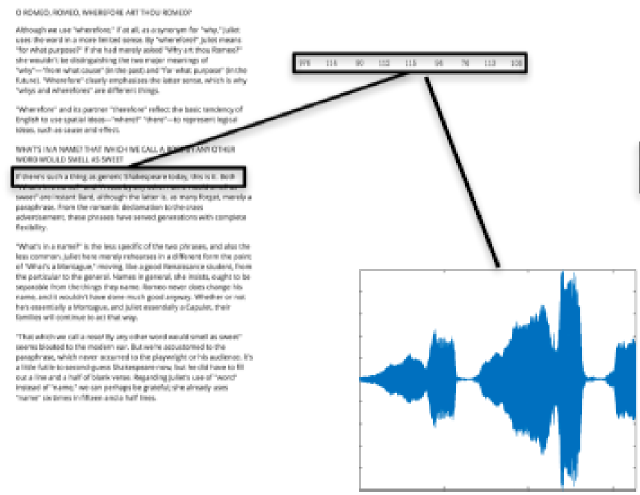


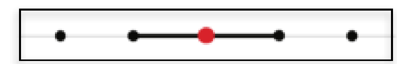
- Convolutional layers
  - Parameters are shared across different "neurons"
  - Huge reduction in computational complexity
    - With respect to Fully-Connected (FC) layers



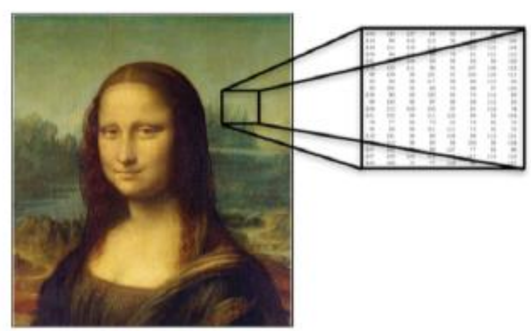
Data domain for CNNs

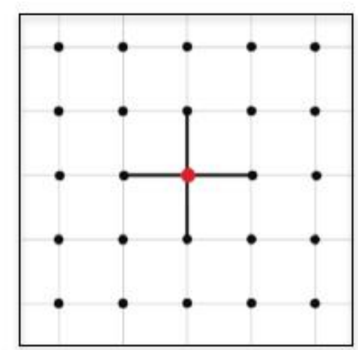
- Domains with regular sampling structures
- 1-D Euclidean domain
  - Sentences, words
  - Sound
- 2-D Euclidean domain
  - Images
- 3-D Euclidean domain
  - Video
  - Volumes





1D grid

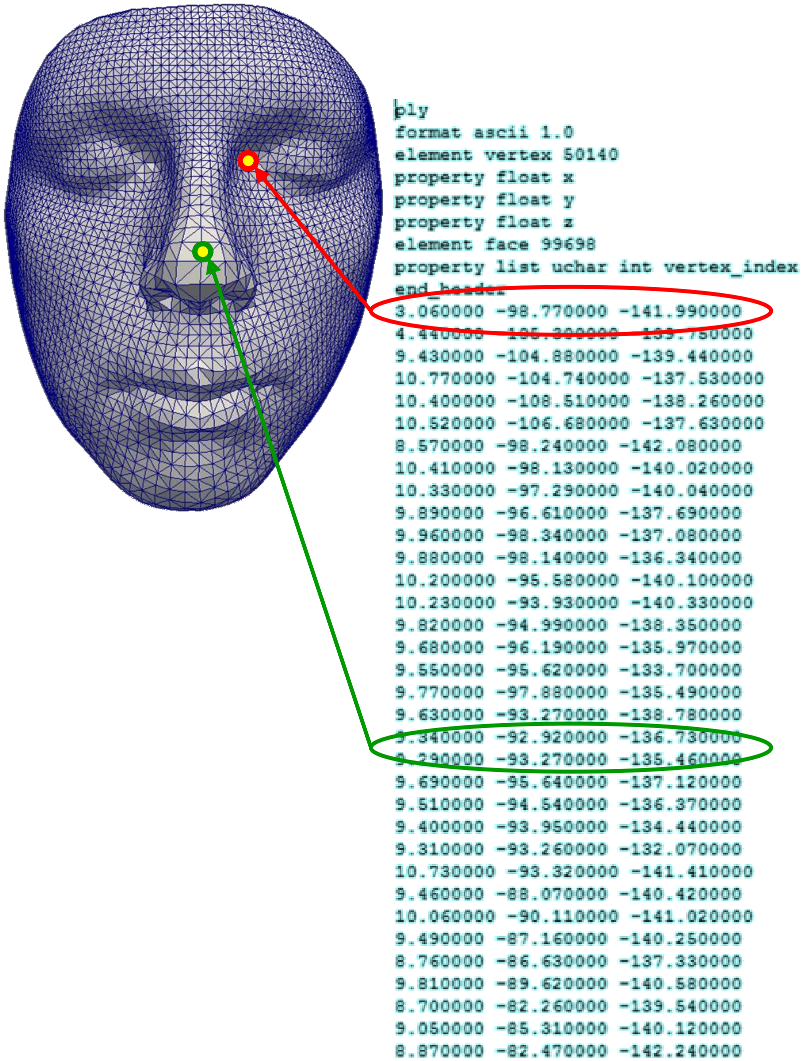


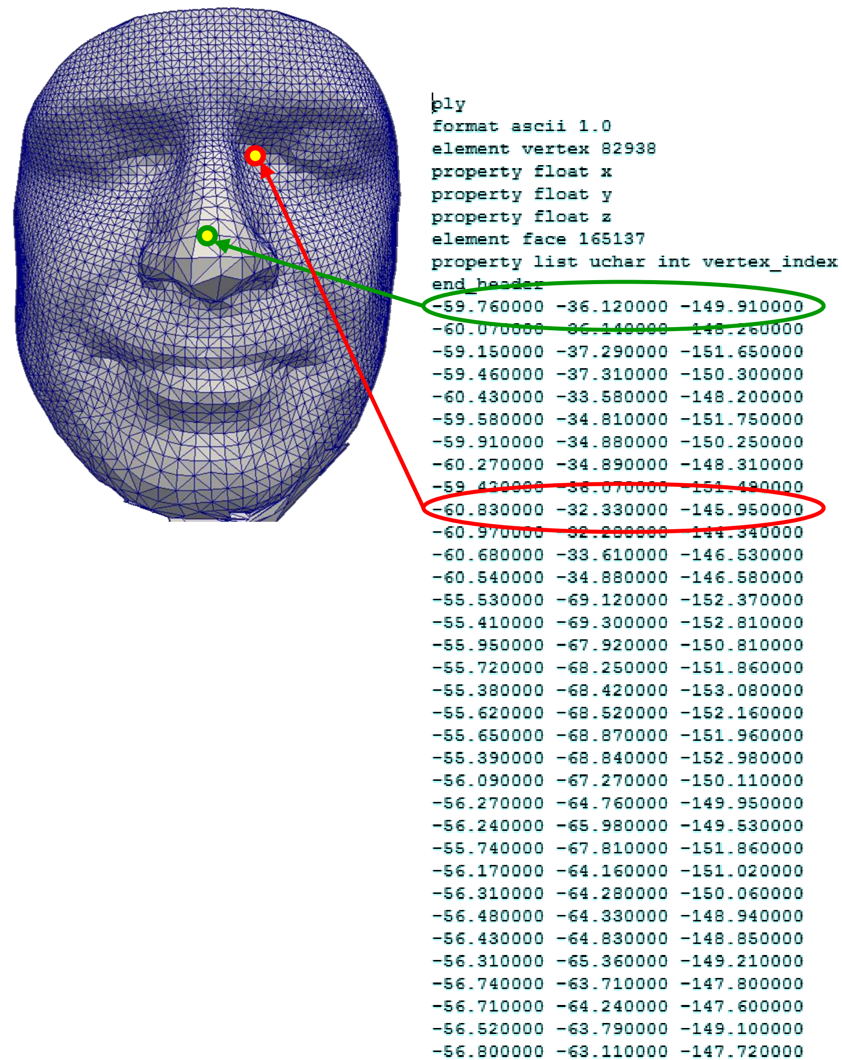


2D grids

How about point clouds?

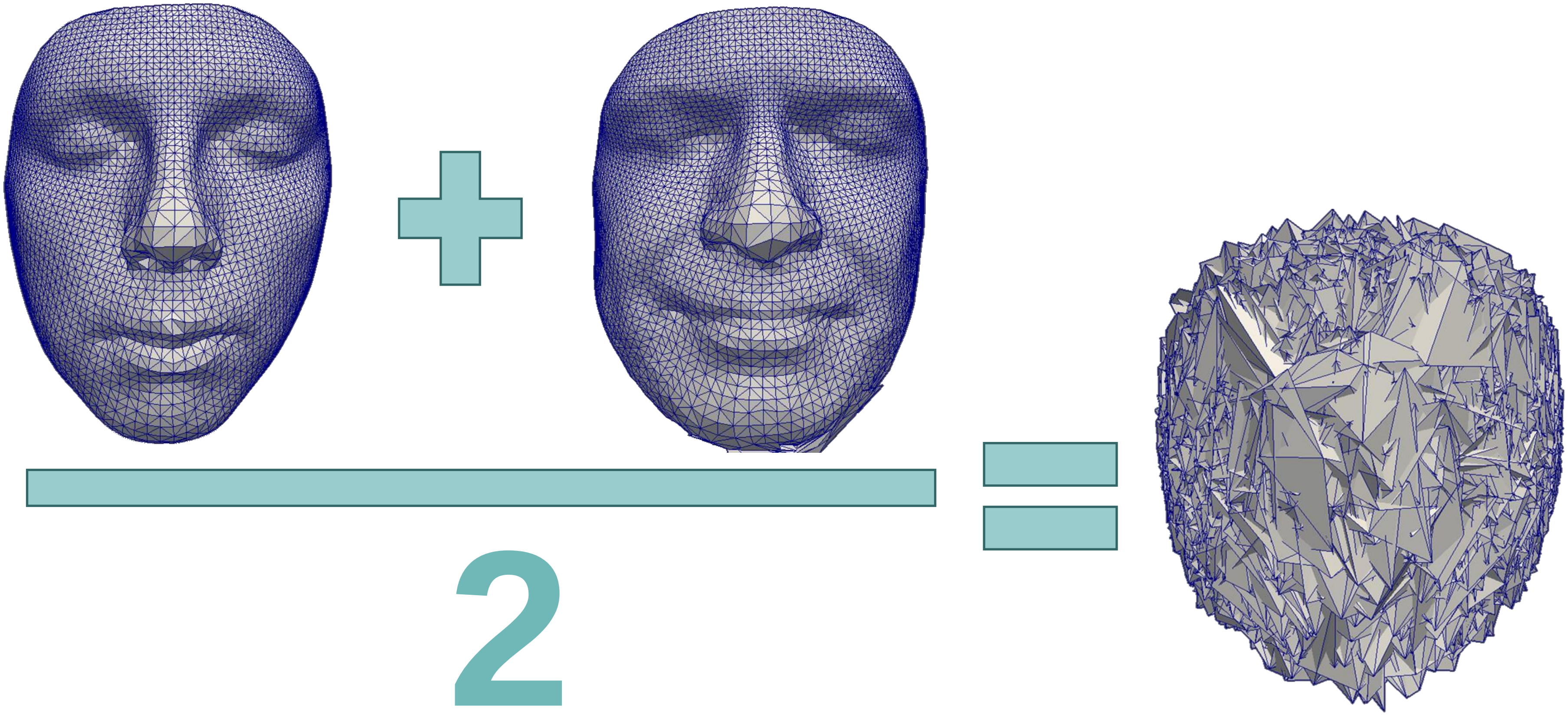
Input points are inherently not ordered





How about point clouds?

Input points are inherently not ordered



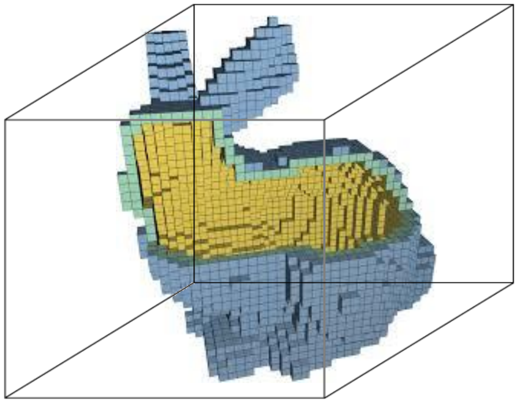
This does not work because surface points are not in correspondence

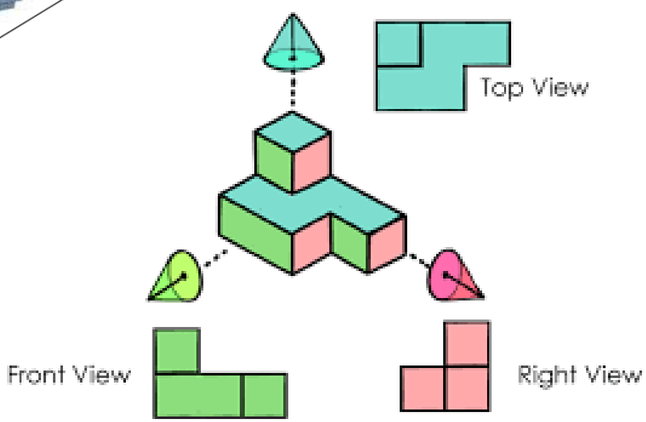
How can we apply DL to point clouds?

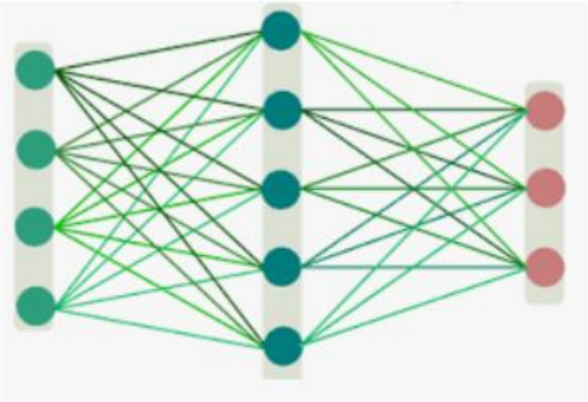
- Transform the representation
  - Voxelization
    - 3D-CNN
  - Projection / rendering
    - 2D-CNN
  - Feature extraction
    - Fully-connected networks
- Networks specifically designed for point clouds
  - PointNet / PointNet++
- Graph Neural Netowkrs
  - Generalization to non-Euclidean sampling grids

DL on point-clouds
Transformed Representations

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

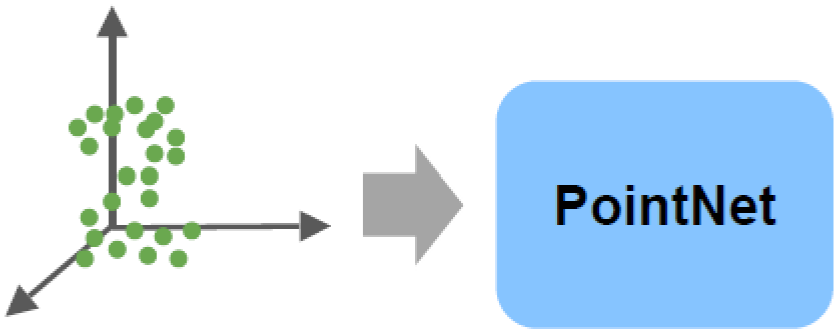


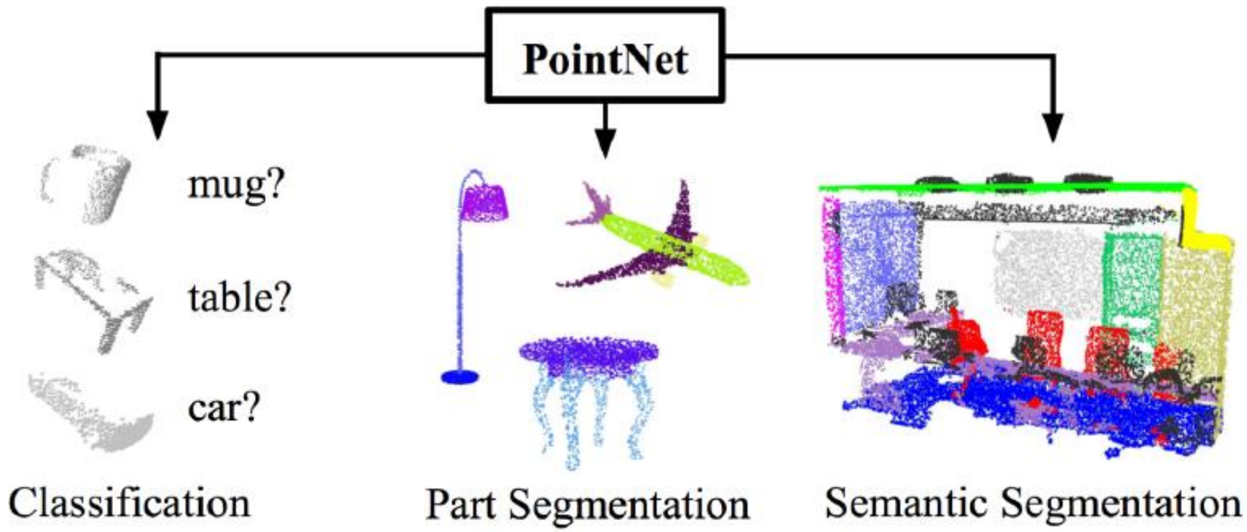




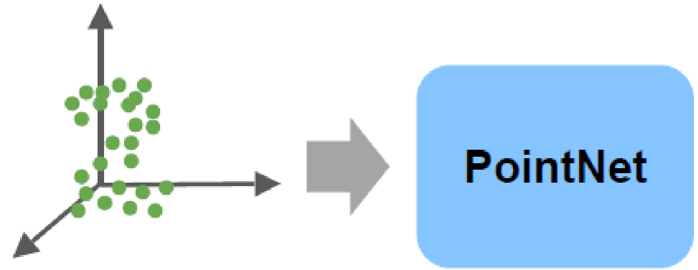
DL on point-clouds: PointNet

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

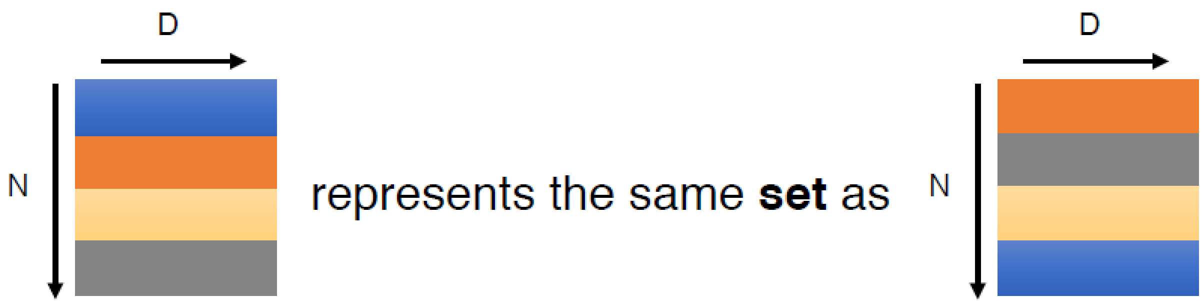




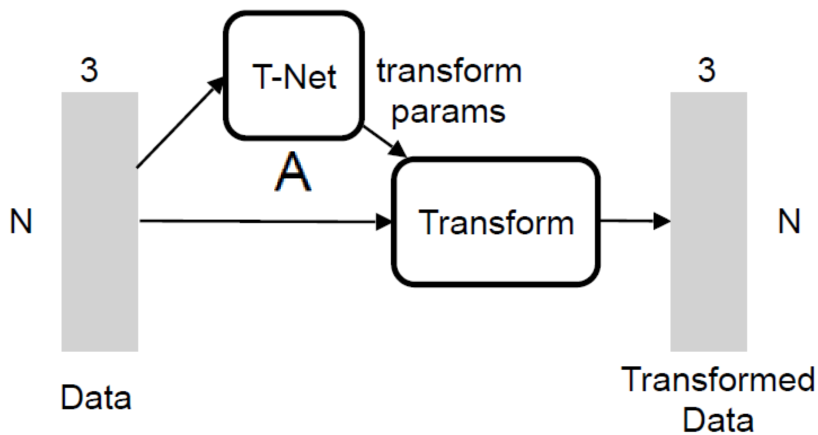
PointNet



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



- Invariance to rigid transformations

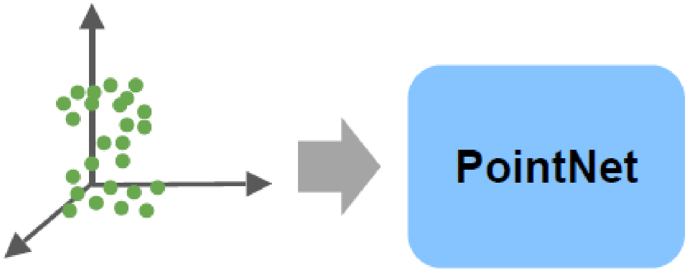


Regularization:

Transform matrix A close to orthogonal:

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

PointNet



- The model should be invariant to permutations

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

  - Examples

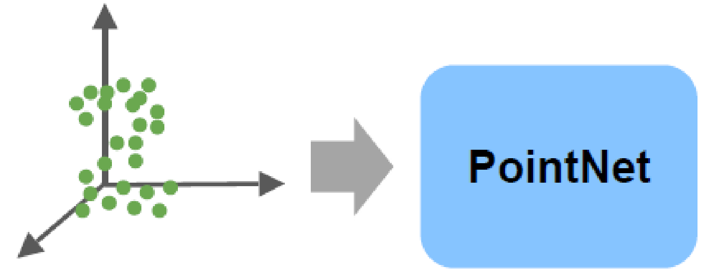
$$f(x_1, x_2, \ldots, x_n) = \max\{x_1, x_2, \ldots, x_n\}$$

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

…

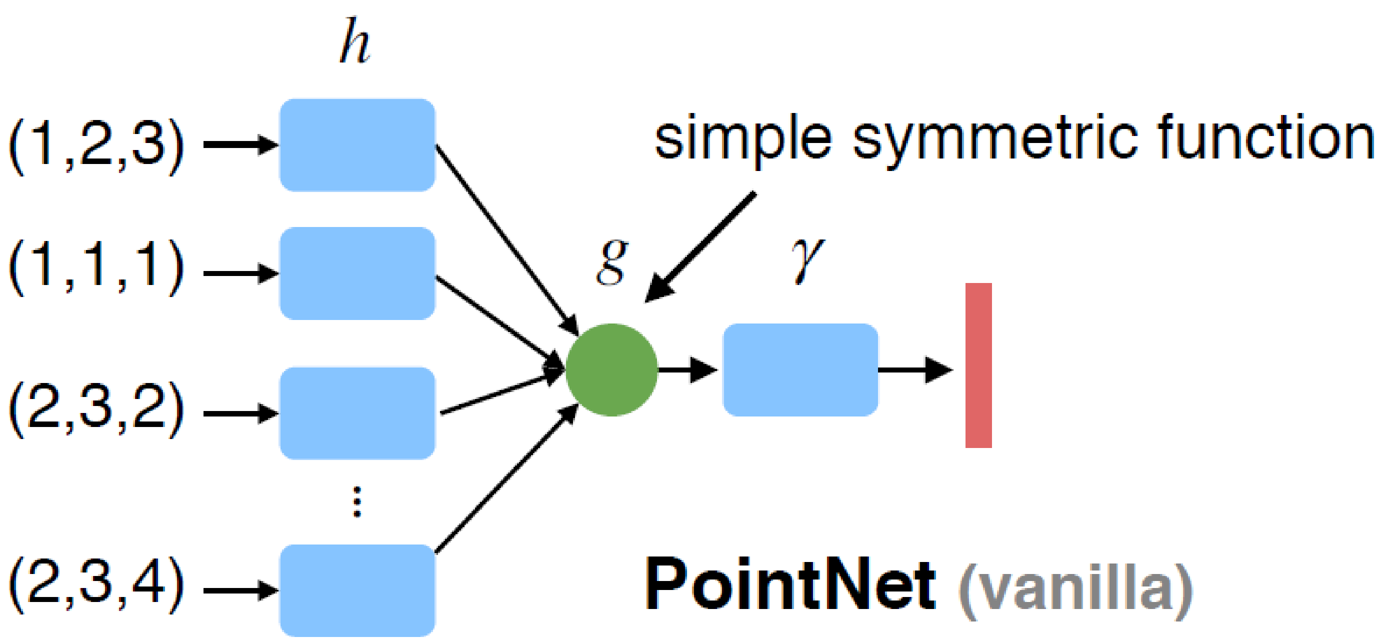
  - Then:
    - How can we construct a family of symmetric functions with neural networks?

PointNet

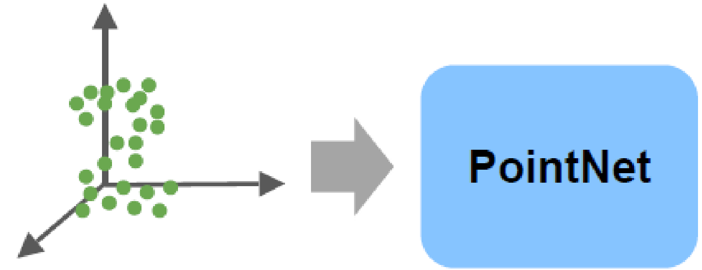


- For the function below
  - f is symmetric if g is symmetric

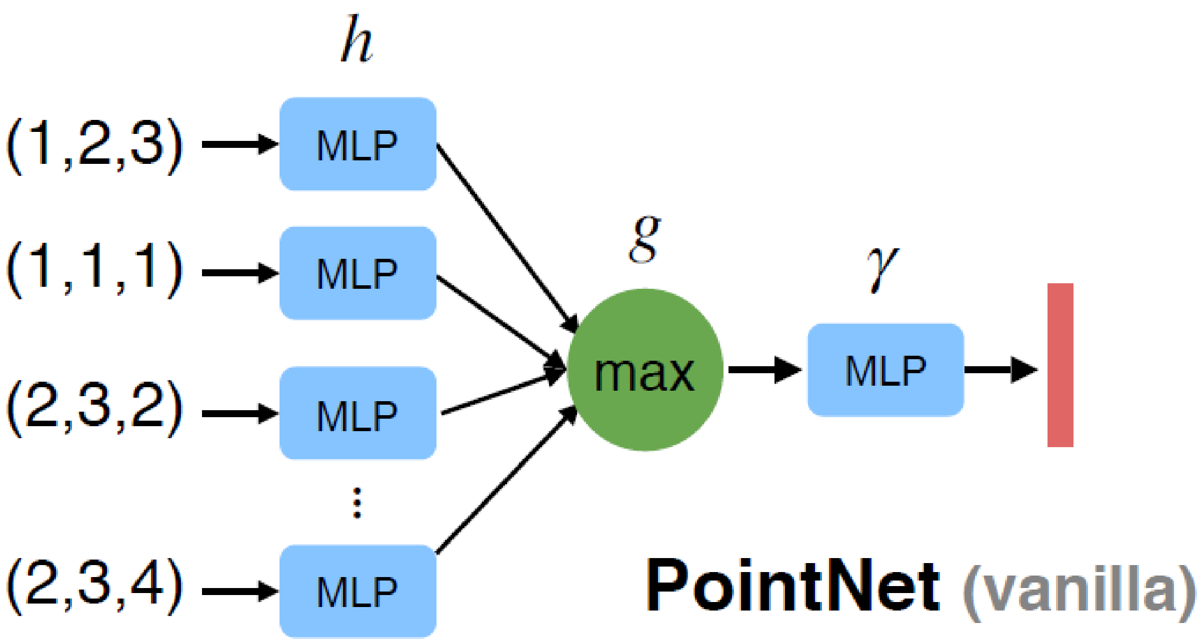
$$f(x_1, x_2, \ldots, x_n) = \gamma \circ g(h(x_1), \ldots, h(x_n))$$



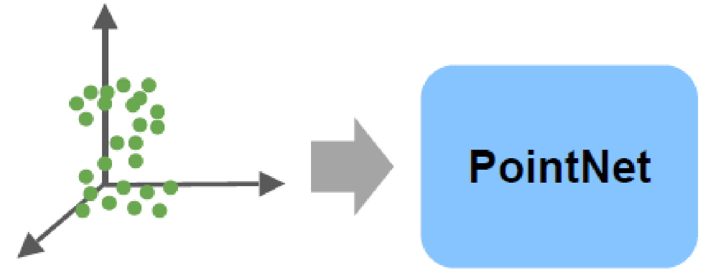
PointNet



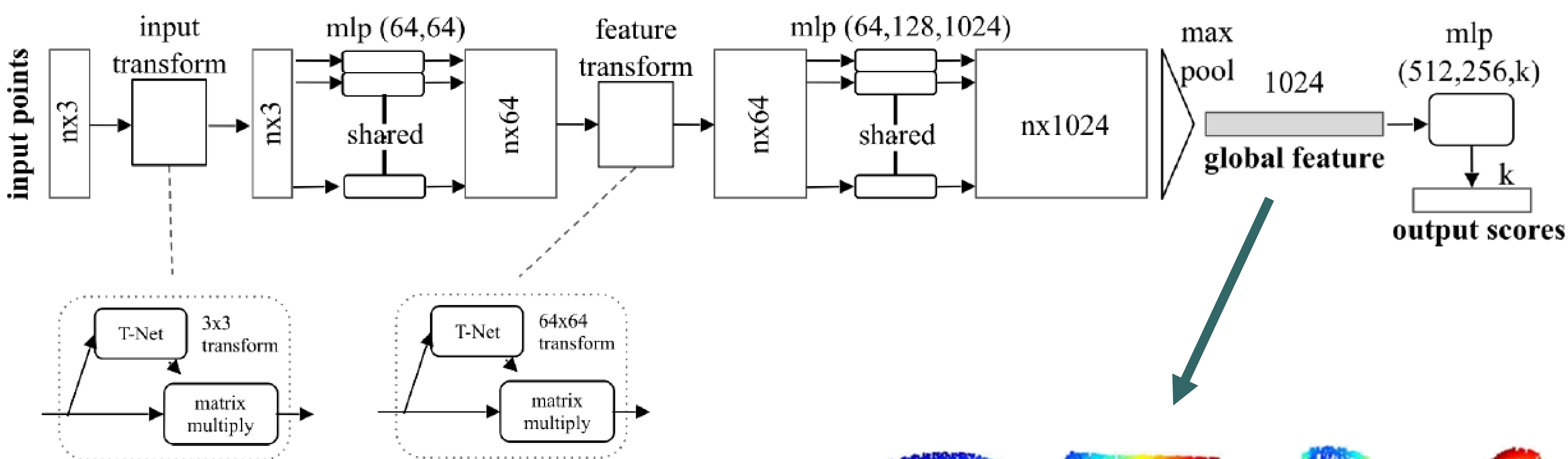
- In PointNet
  - g was chosen to be the MAX function
  - γ was chosen to be a Multi-Layer Perceptron (MLP)

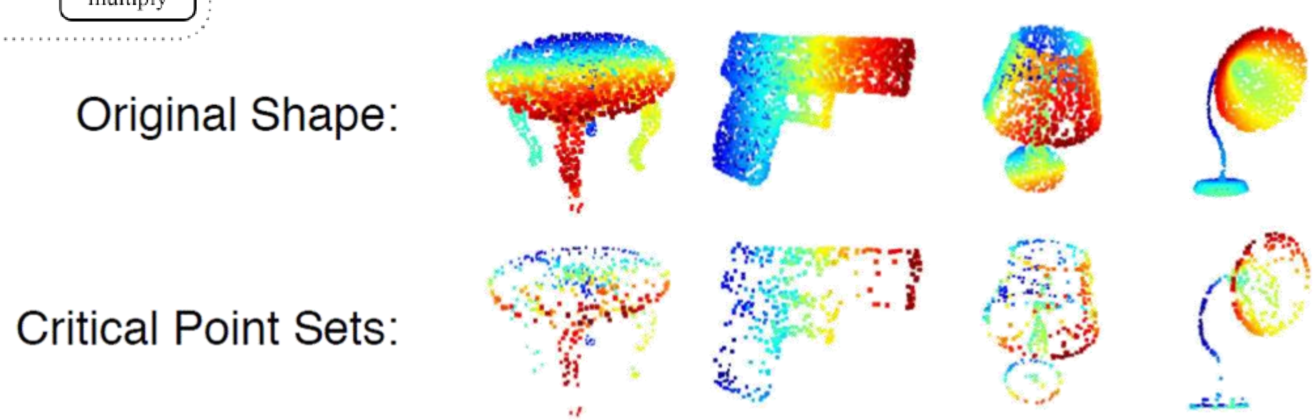


PointNet



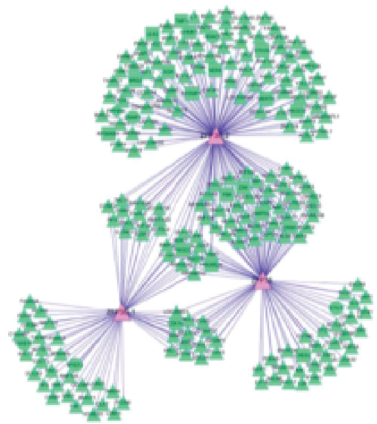
- PointNet architecture



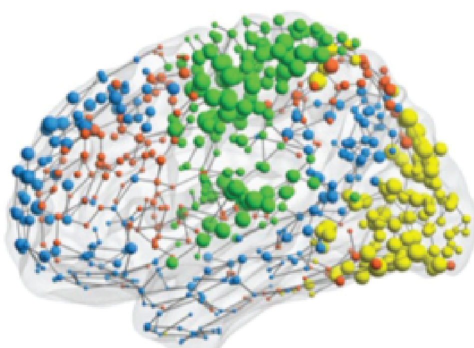


Non-Euclidean data

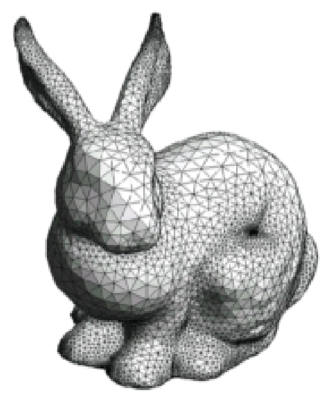
Social networks



Regulatory networks

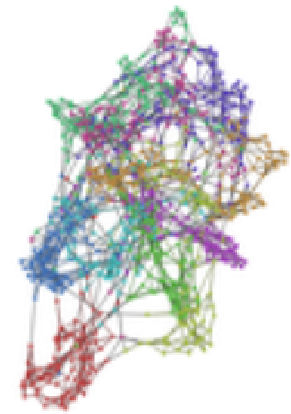


Functional networks



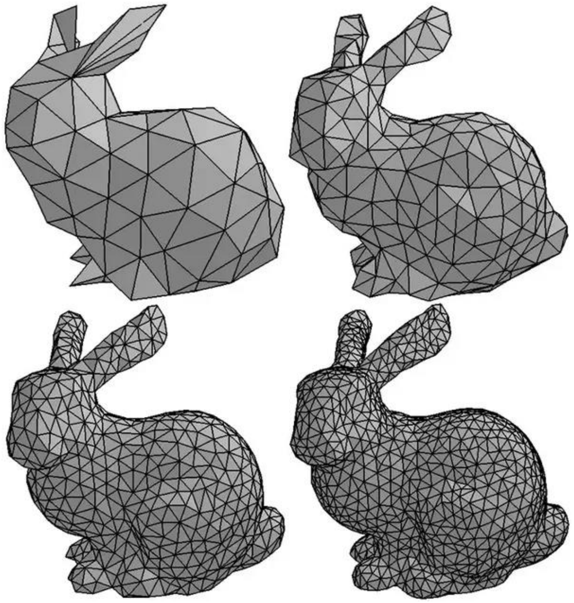
3D shapes

=



Graphs/ Networks

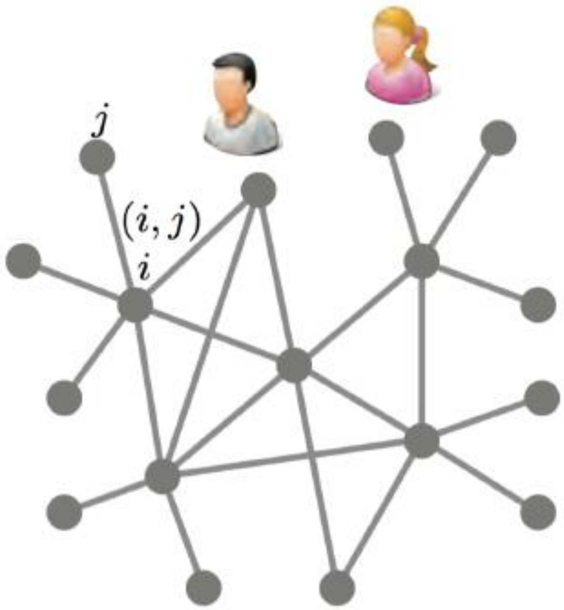
Non-Euclidean data



- Can be handled generically by using 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 $\mathbf{f} = (f_1, \ldots, f_n)$

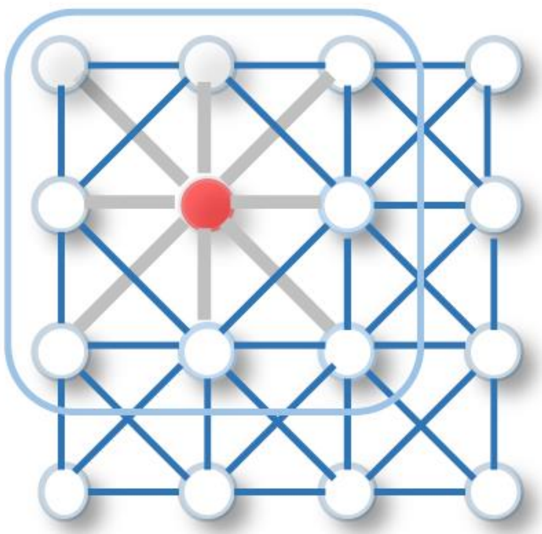


DL in Non-Euclidean data

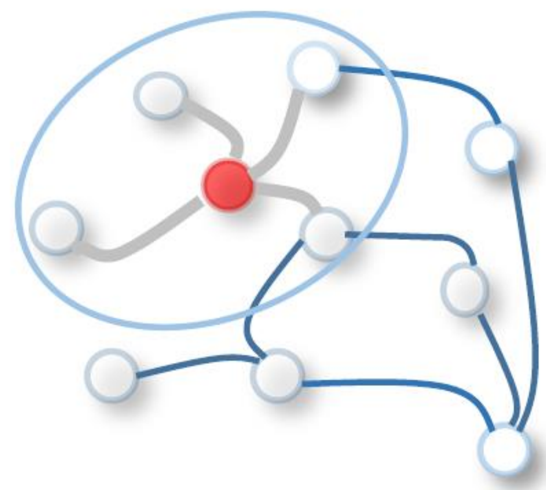
- Assumptions
  - Non-Euclidean data are
    - Locally stationary
    - Compositional
- Challenges
  - How can we extend CNNs to graph-structure data?
    - How can we extend convolutions to graphs?
  - How to define compositionality on graphs?
    - Convolution
    - Pooling
  - How can we make the 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

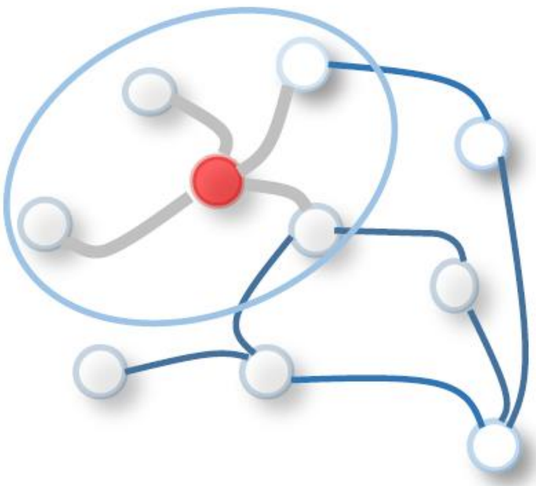


Non-Euclidean neighborhood

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

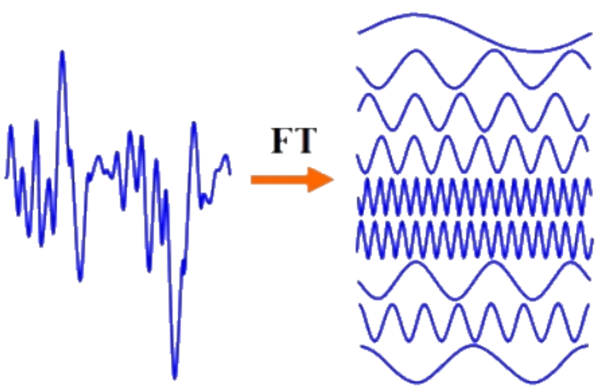
DL in Non-Euclidean data

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

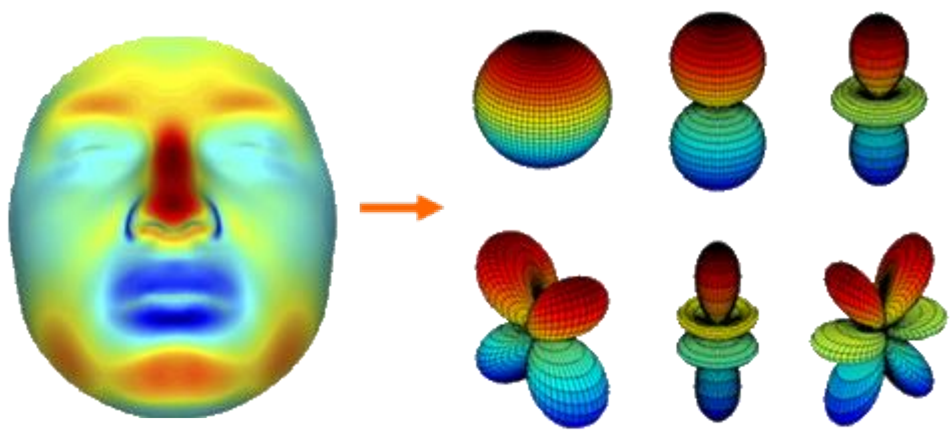
- 3 main solutions have been proposed
  - DL extensions based on RNNs
    - Message passing between neighboring nodes
  - DL extensions based on CNNs
    - Convolutions in the spectral domain
    - Convolutions directly on the graph

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



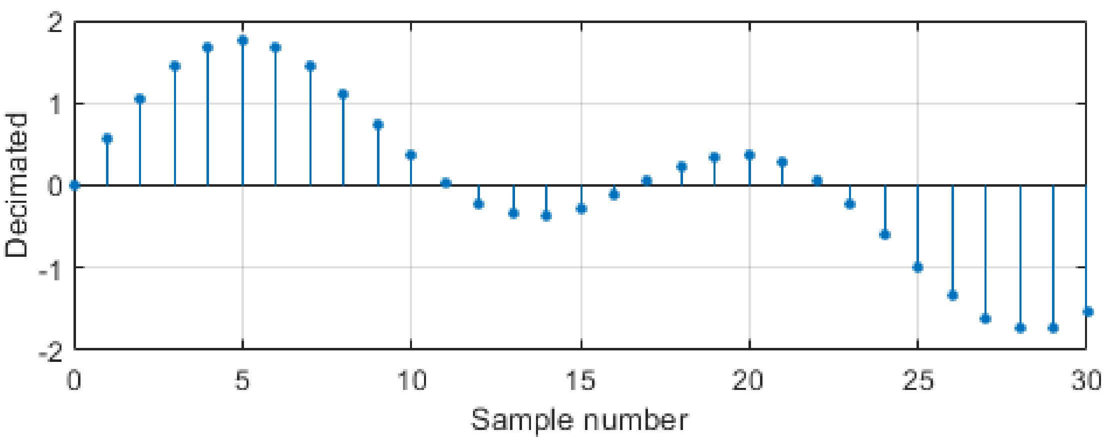
Signal in 1D



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

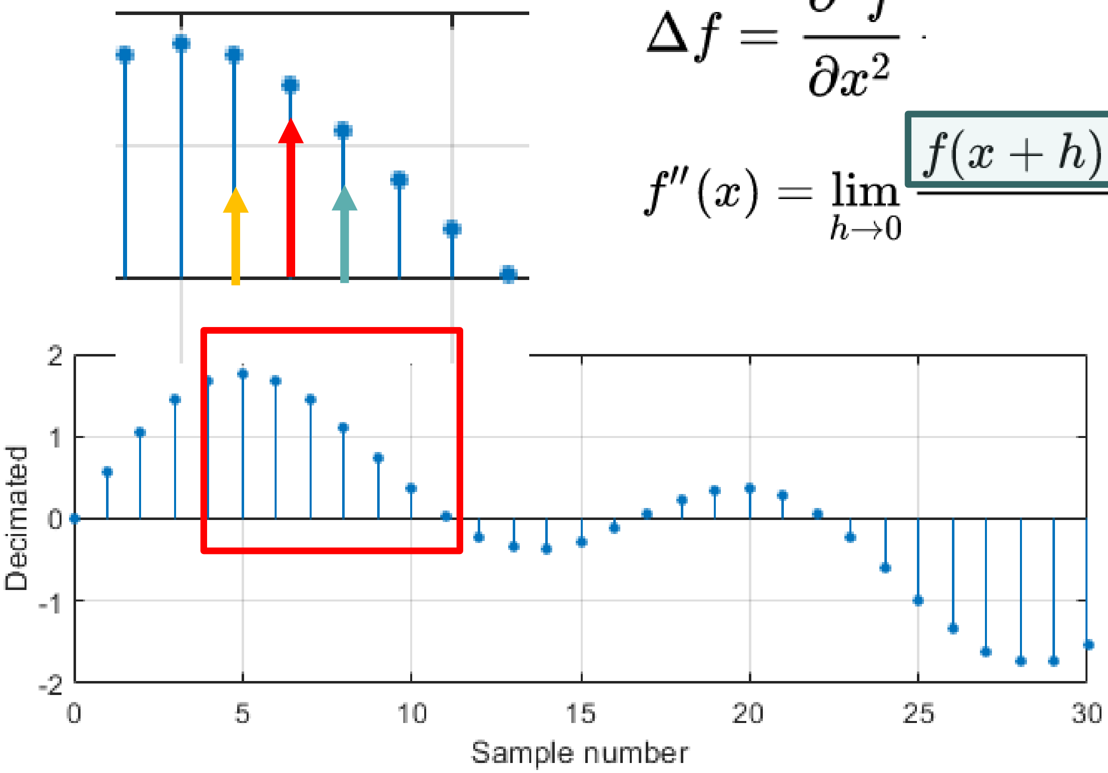


$$\Delta x = -K\,x$$

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

Spectral methods on graphs

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



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

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

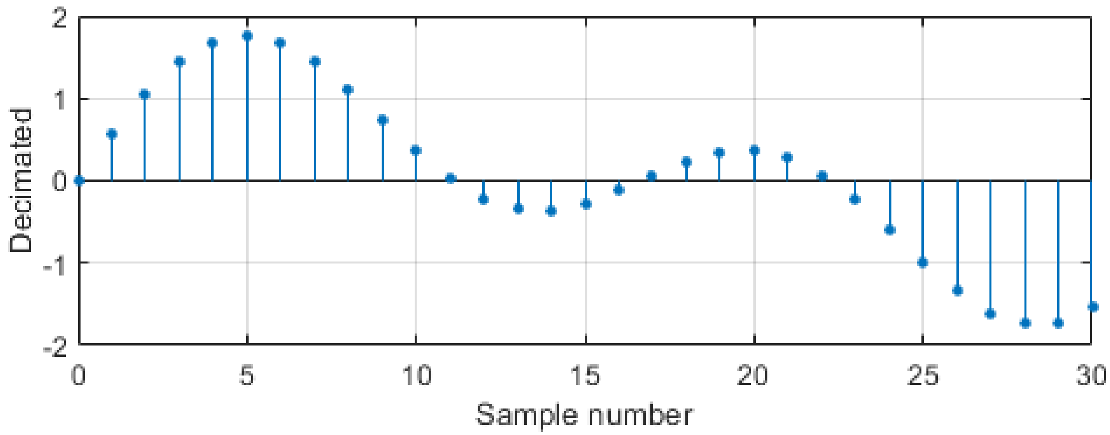
$$\Delta x = -K\,x$$

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

Spectral methods on graphs

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

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

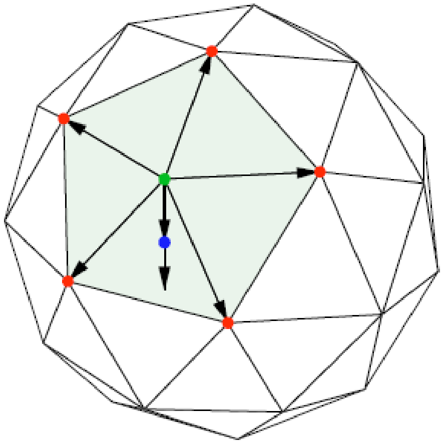


$$\Delta x = -K\,x$$

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

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



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

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

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

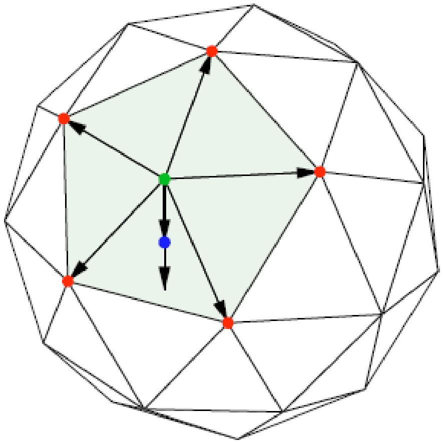
1-D case

$$\boldsymbol{\Delta x} = -K\boldsymbol{x}$$

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

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
    - The adjacency can be further generalized to edge weights **W**



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

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

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

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

Spectral methods on graphs

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

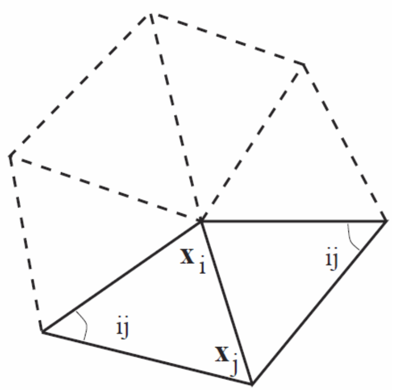
$$\mathbf{\Delta} = \mathbf{D} - \mathbf{A}$$

- Normalized Laplacian

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

- Laplace-Beltrami Operator
  - Geometric Laplacian

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



Spectral methods on graphs

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

$$\boldsymbol{\Delta}\phi_k = \lambda_k \phi_k, \quad k = 1, 2, \ldots \qquad \boldsymbol{\Delta} = \boldsymbol{\Phi}^T \boldsymbol{\Lambda} \boldsymbol{\Phi}$$

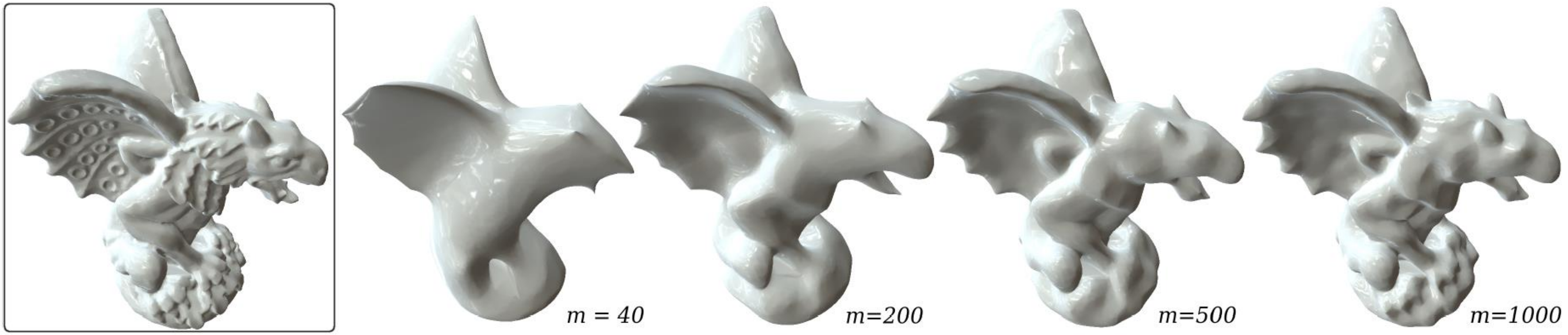
  - Eigenvectors are real and orthonormal

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

  - The eigenvectors of the Lalpacian 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)

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

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

Fourier basis

Spectral convolution

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

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

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

- Where the Fourier basis comes from the Laplacian

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

- Therefore

$$\mathbf{f} \star \mathbf{g} \quad = \quad \overset{\mathscr{F}^{-1}}{\boxed{\mathbf{\Phi}}}(\underbrace{\overset{\mathscr{F}}{\boxed{\mathbf{\Phi}^\top}}\mathbf{g}}_{\hat{g}_\theta(\Lambda) = \mathrm{diag}(\theta)} \circ \overset{\mathscr{F}}{\boxed{\mathbf{\Phi}^\top}}\mathbf{f})$$

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

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

Spectral convolution

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

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

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

- Where the Fourier basis comes from the Laplacian

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

- Theretore

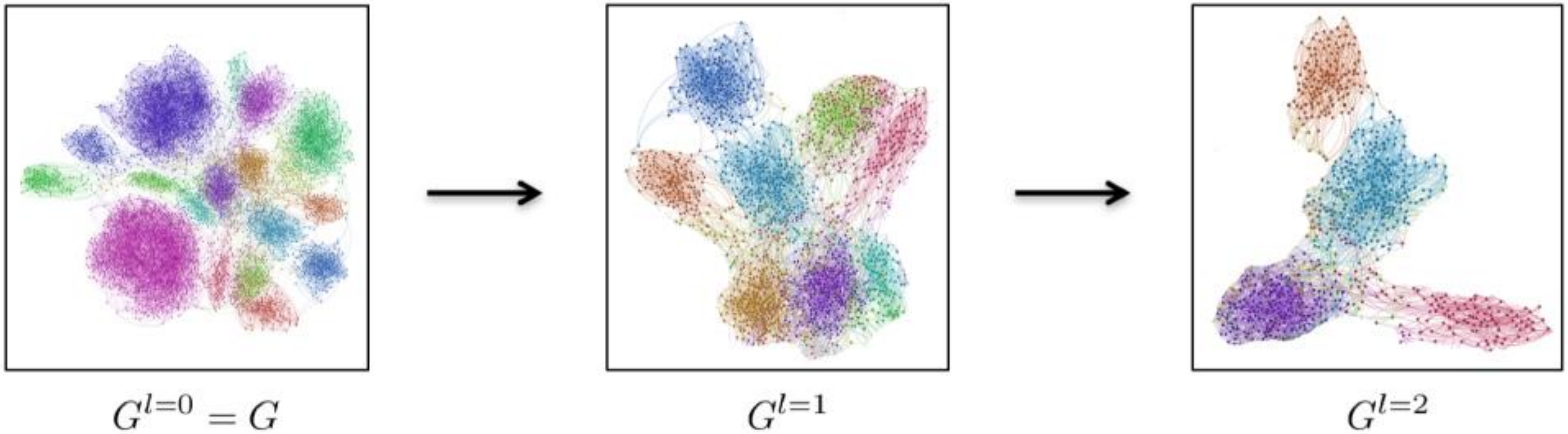
$$\mathbf{f} \star \mathbf{g} \quad = \quad \mathbf{\Phi}\,(\underbrace{\mathbf{\Phi}^\top \mathbf{g}}_{\hat{g}_\theta(\Lambda) = \mathrm{diag}(\theta)} \circ \mathbf{\Phi}^\top \mathbf{f})$$

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

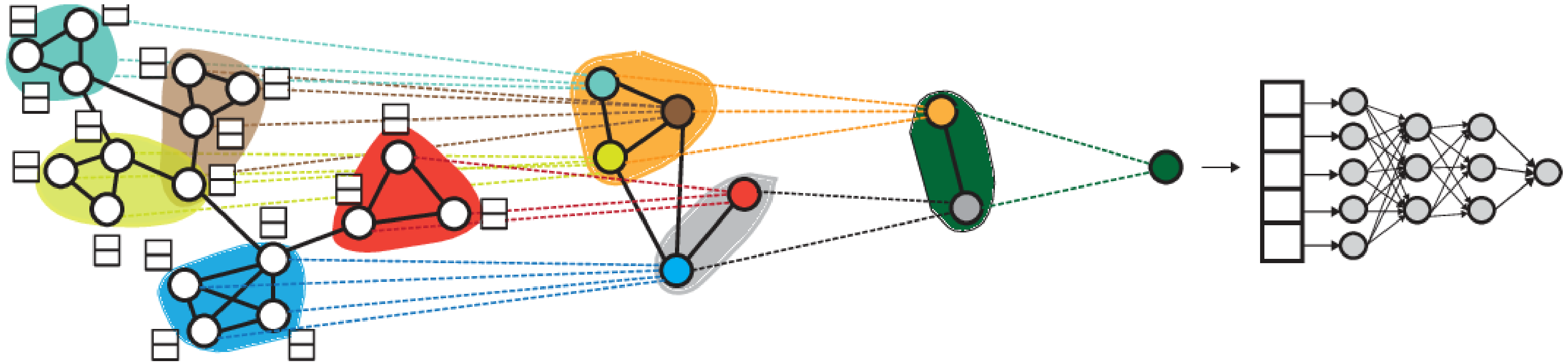
$\theta \in \mathbb{R}^n$ is a vector of Fourier coefficients

Graph pooling

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



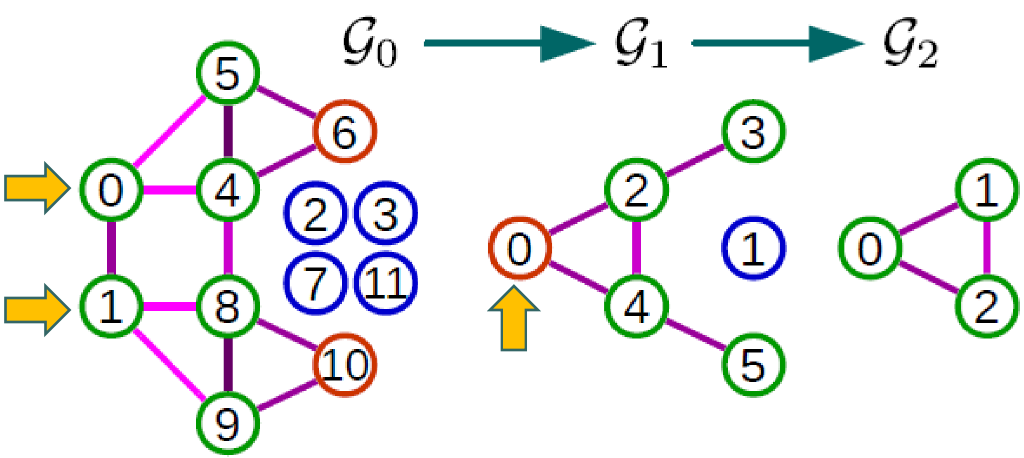
Graph pooling

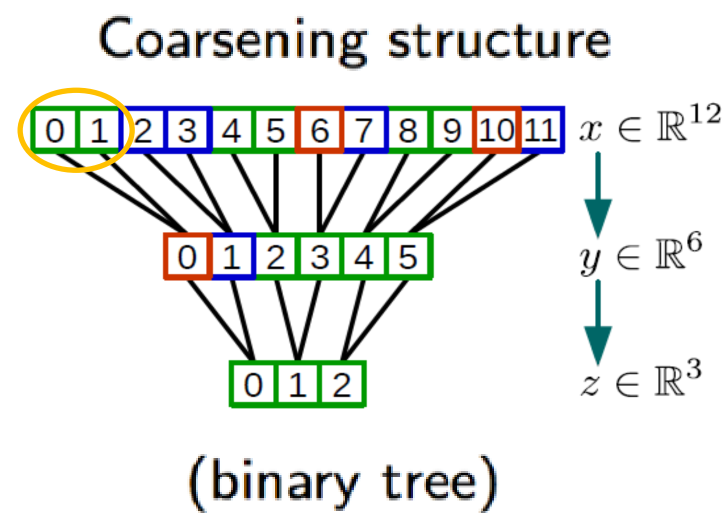


- Approximate solutions
  - Not guaranteed to be optimal
  - But often suffice for our needs
- Structured pooling
  - Arrangement of the node indexing
  - Adjacent nodes are hierarchically merged

Graph pooling

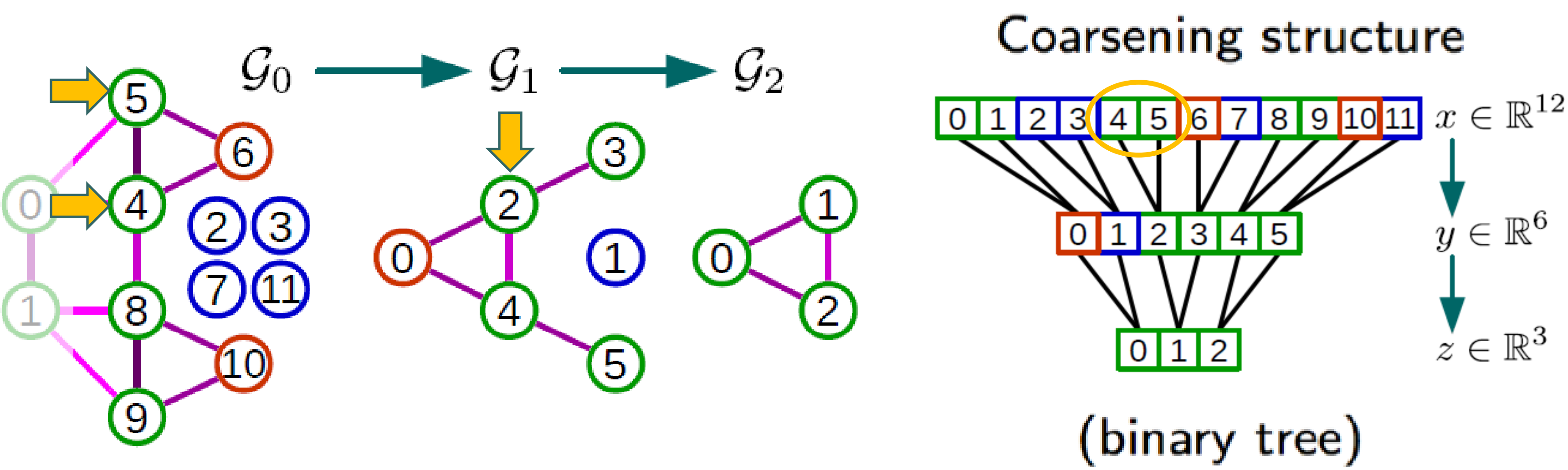
- Structured pooling
  - Arrangement of the node indexing
  - Adjacent nodes are hierarchically merged
    - Requires adding “ghost” nodes
  - As efficient as a 1D-Euclidean grid pooling





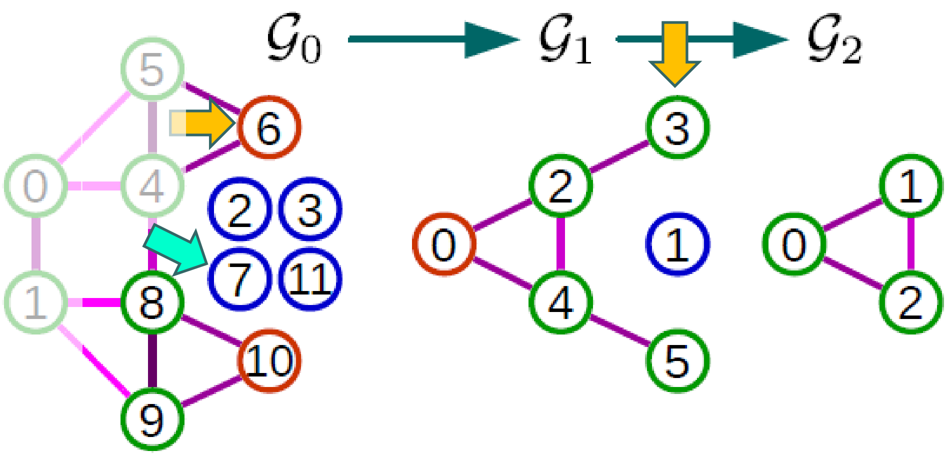
Graph pooling

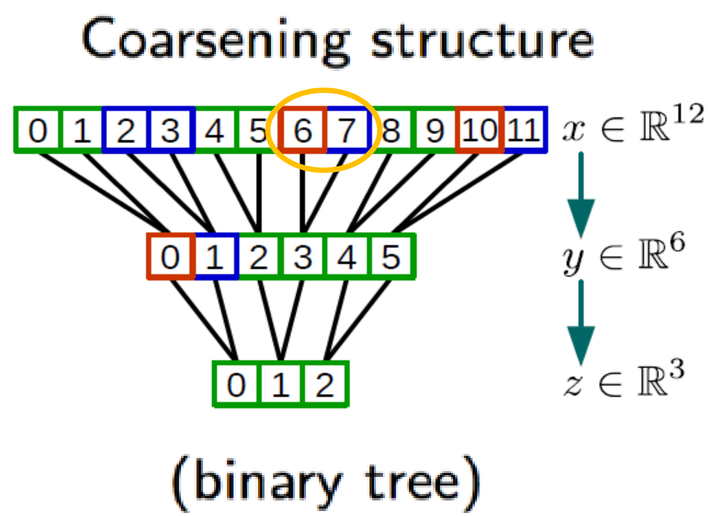
- Structured pooling
  - Arrangement of the node indexing
  - Adjacent nodes are hierarchically merged
    - Requires adding "ghost" nodes
  - As efficient as a 1D-Euclidean grid pooling



Graph pooling

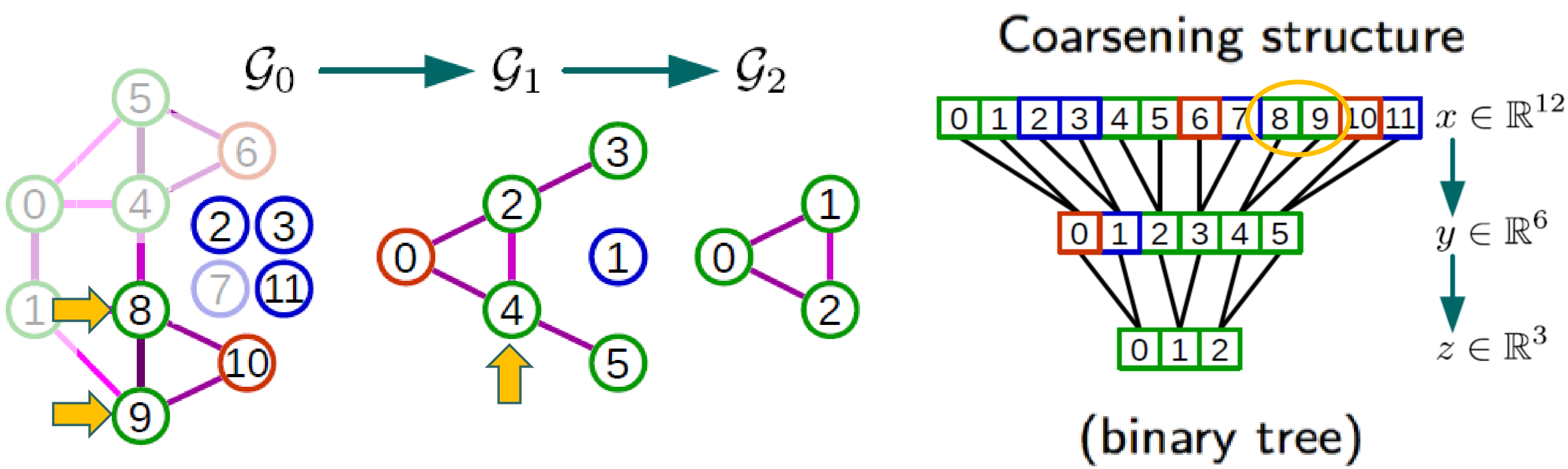
- Structured pooling
  - Arrangement of the node indexing
  - Adjacent nodes are hierarchically merged
    - Requires adding “ghost” nodes
  - As efficient as a 1D-Euclidean grid pooling





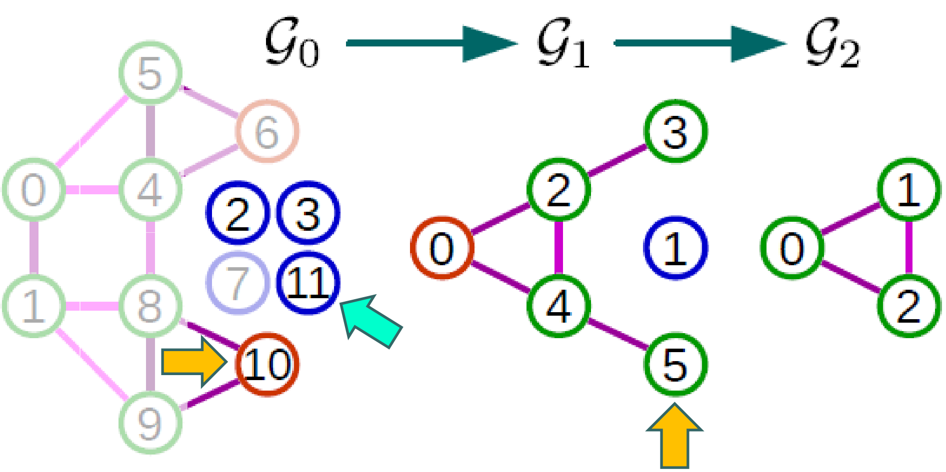
Graph pooling

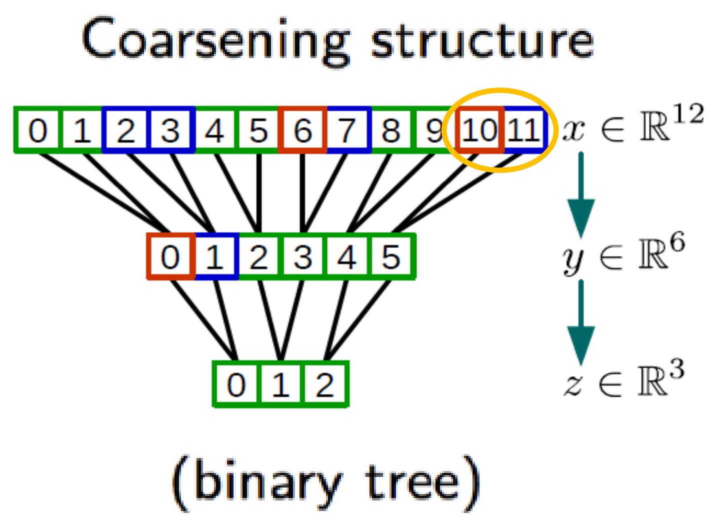
- Structured pooling
  - Arrangement of the node indexing
  - Adjacent nodes are hierarchically merged
    - Requires adding “ghost” nodes
  - As efficient as a 1D-Euclidean grid pooling



Graph pooling

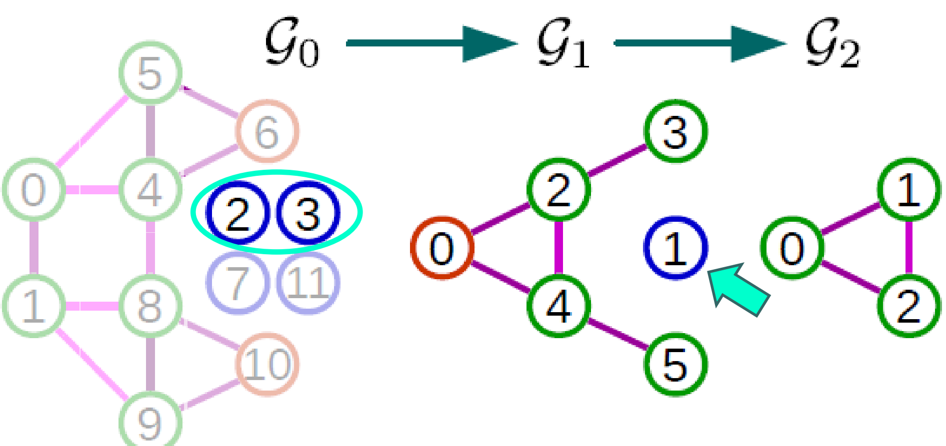
- Structured pooling
  - Arrangement of the node indexing
  - Adjacent nodes are hierarchically merged
    - Requires adding “ghost” nodes
  - As efficient as a 1D-Euclidean grid pooling

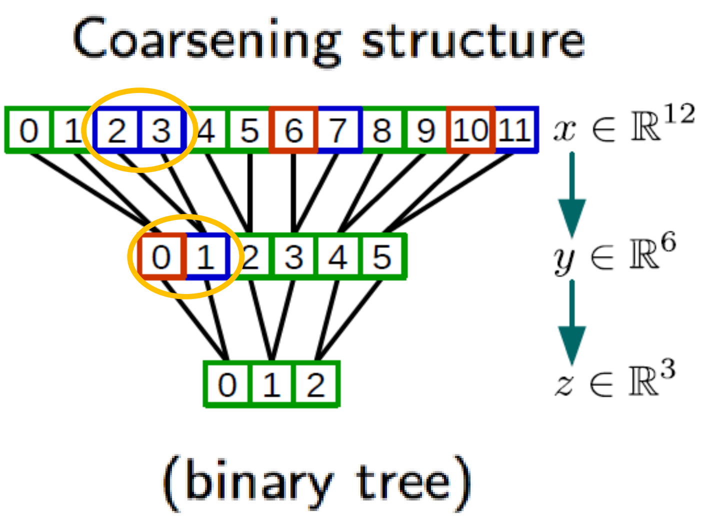




Graph pooling

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

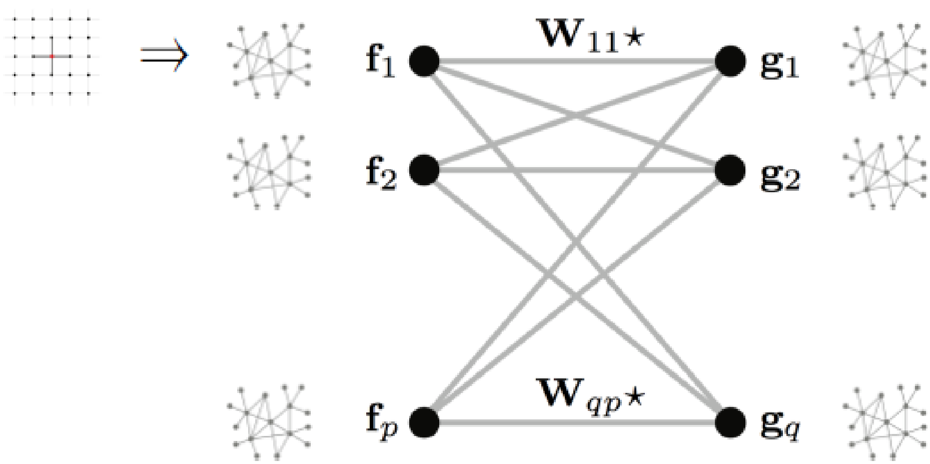




Vanilla graph CNNs

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

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

Vanilla spectral graph CNNs

- The convolutional layer in the spatial domain

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

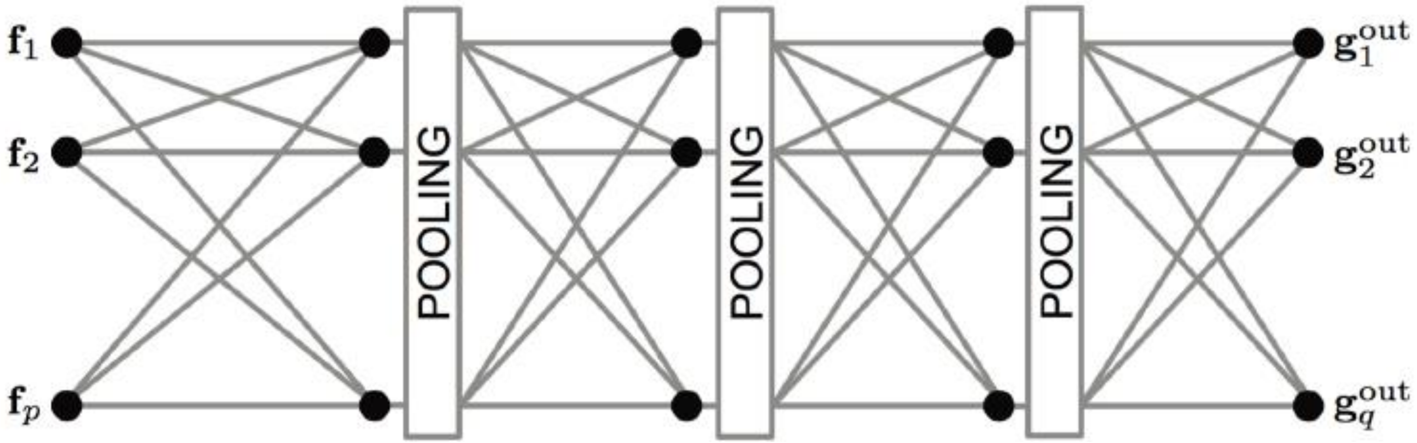
- Can be replaced by a filter expressed in the spectral domain

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

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

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

Graph Compositional layers



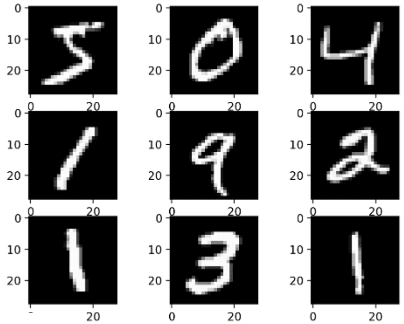
$$\begin{aligned} \mathbf{f}_l &= l\ l\text{-th data feature on graphs, } \dim(\mathbf{f}_l) = n \times 1 \\ \mathbf{g}_l^{(k)} &= l\ l\text{-th feature map, } \dim(\mathbf{g}_l) = n \times 1 \end{aligned}$$

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

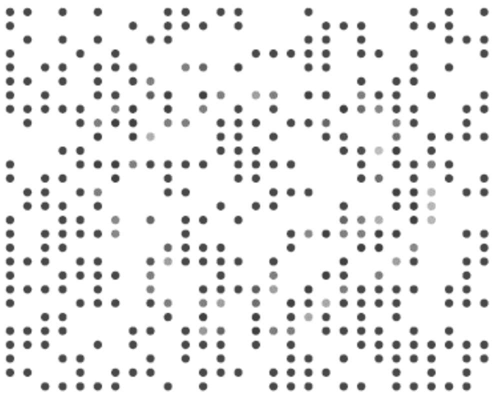
Activation, e.g. $\quad \xi(x) = \max\{x, 0\}$ rectified linear unit (ReLU)

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

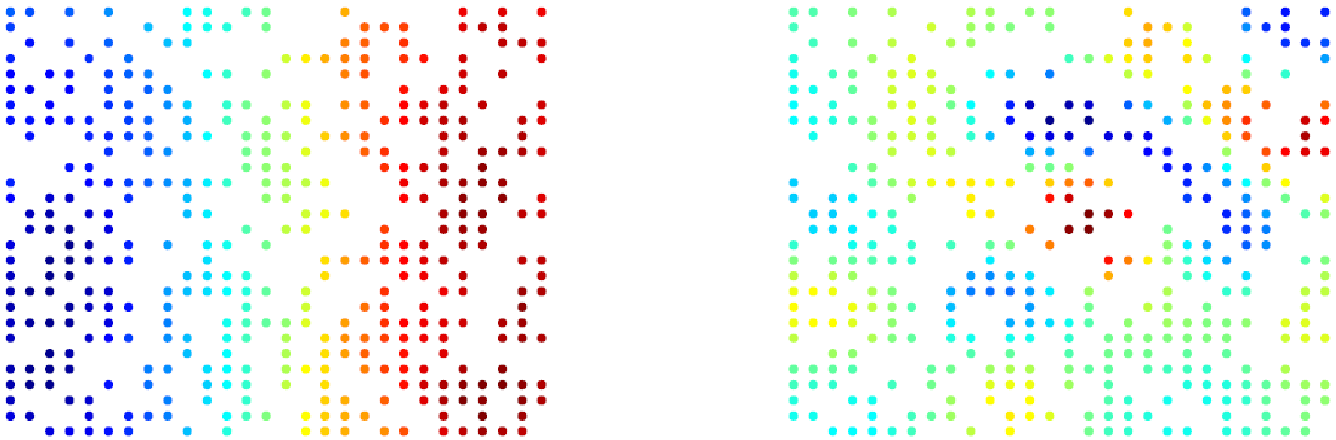
Graph CNNs on a synthesized graph



- MNIST digits database
  - Images were sub-sampled
    - 400 points
    - Non-regular graph



  - Example of the first two eigenvectors of the Laplacian

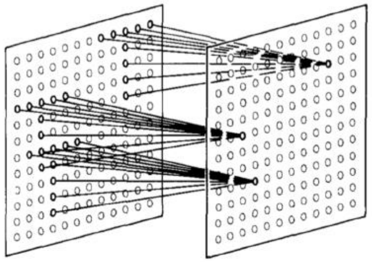


Spectral domain issues

- Direct operation in spectral domain

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

  - 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 invers 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

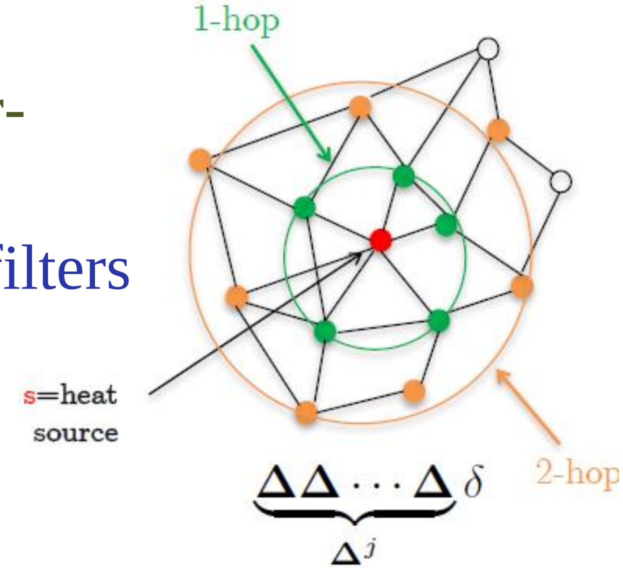
$$w_{\boldsymbol{\alpha}}(\lambda) = \sum_{j=0}^{r} \alpha_j \lambda^j$$

ChebNet

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

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

- 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

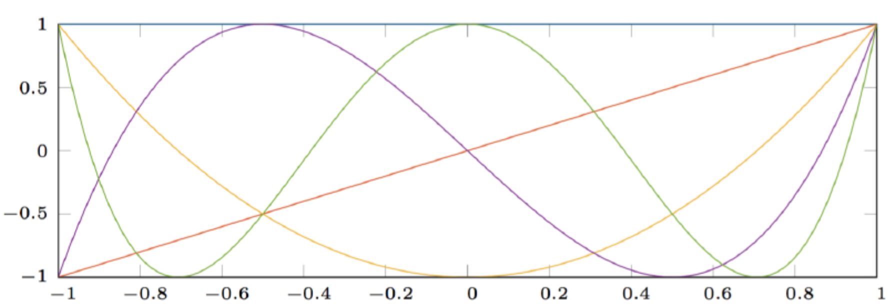
- Chebynet
  - Spectral GCNN $$\mathbf{g}_l^{(k)} = \xi\left(\sum_{l'=1}^{q^{(k-1)}} \mathbf{\Phi}\mathbf{W}_{l,l'}^{(k)}\mathbf{\Phi}^\top \mathbf{g}_{l'}^{(k-1)}\right)$$
  - Spectral polynomials $$\mathbf{g}_l^{(k)} = \xi\left(\sum_{l'=1}^{q^{(k-1)}} w_{\boldsymbol{\alpha}}^{(k)}(\mathbf{\Delta})\,\mathbf{g}_{l'}^{(k-1)}\right)$$
- 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

6K29VN

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

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



T_0, T_1, T_2, T_3

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

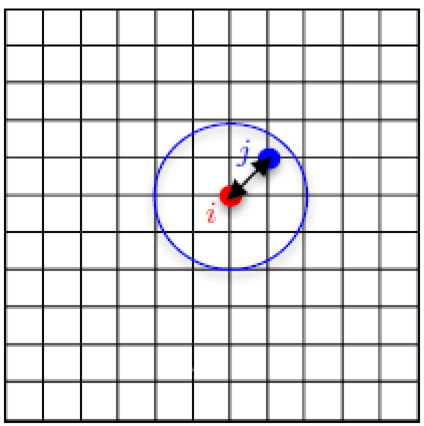
$$T_0(\tilde{\lambda}) = 1, \quad T_1(\tilde{\lambda}) = \tilde{\lambda}$$

Experiments in a Euclidean grid

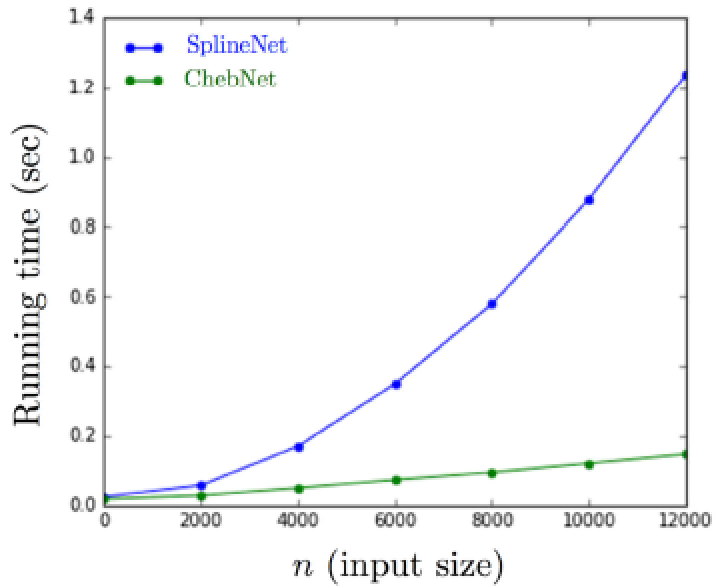
- Digit recognition
  - MNIST database

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

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

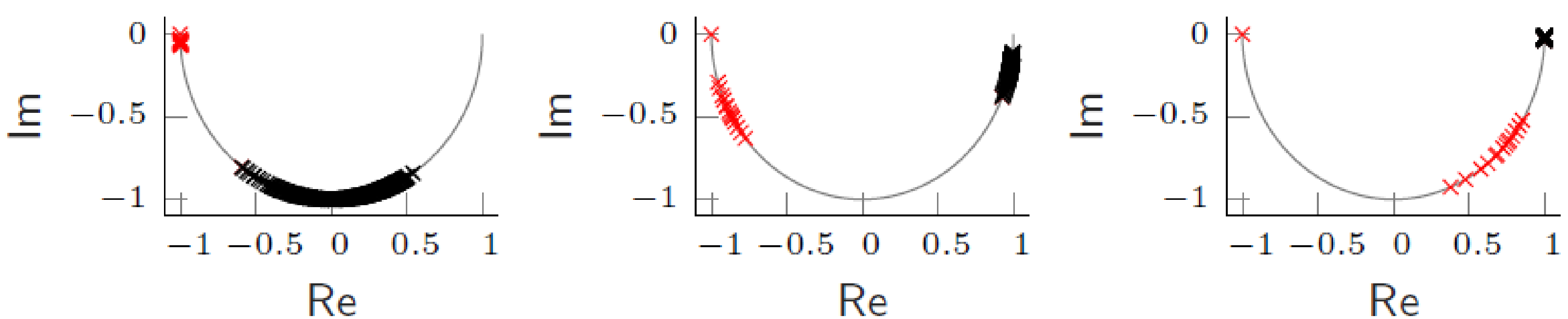
Beyond ChebNet I: CayleyNet

- 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

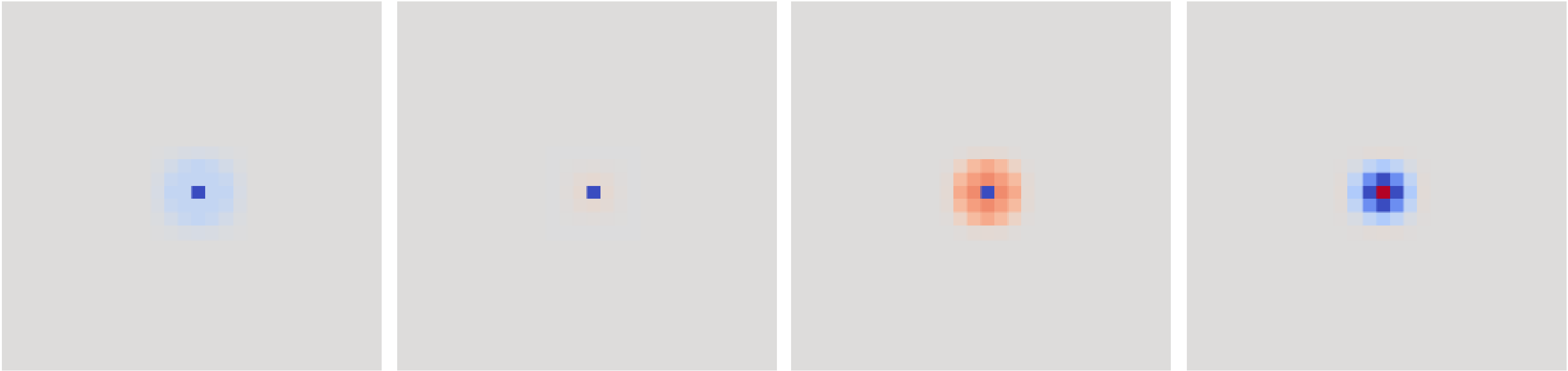
- Cayley polynomials of order r
  - Are a family of real-valued rational functions
  - With complex coefficients

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

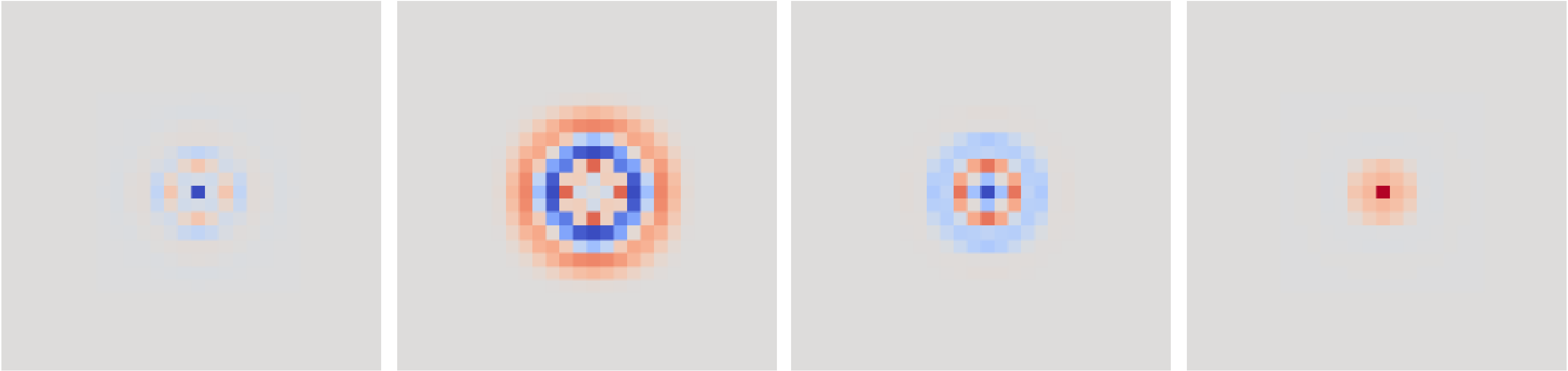
$$= c_0 + \sum_{j=1}^{r} c_j C(h\mathbf{\Delta})^j + \sum_{j=1}^{r} \bar{c_j} C(h\mathbf{\Delta})^{-j}$$

ChebNet vs CayleyNet on a Euclidean grid

- Chebyshev filters of order 3



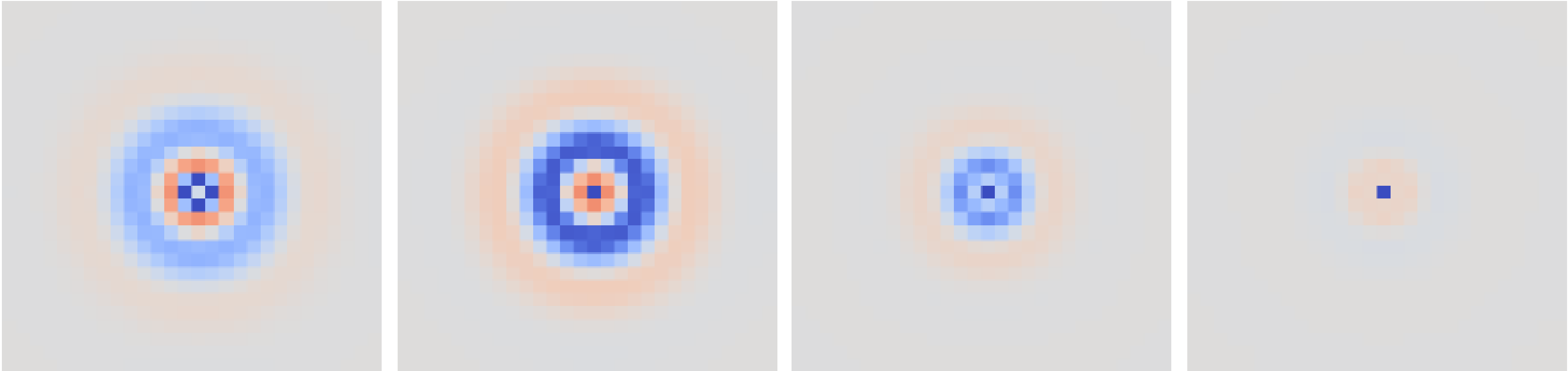
- Chebyshev filters of order 7



ChebNet vs CayleyNet on a Euclidean grid

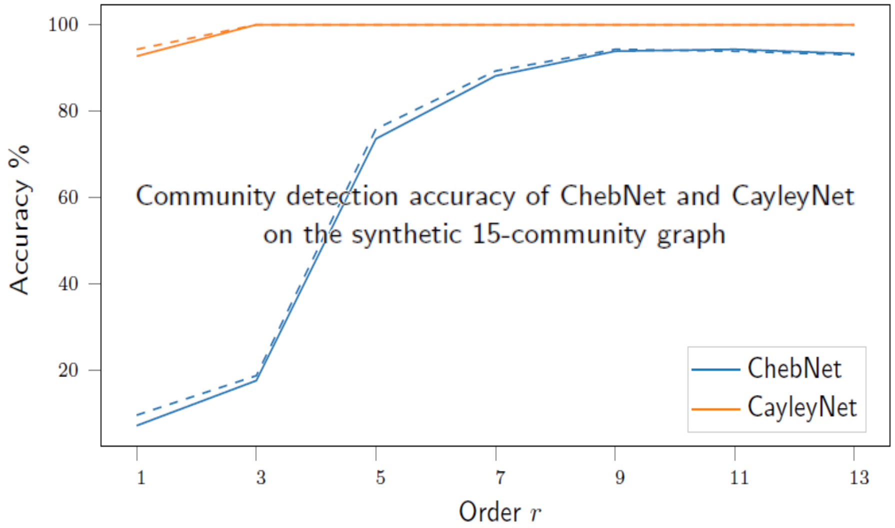
- Chebyshev filters of order 3

- 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 $O(n)$

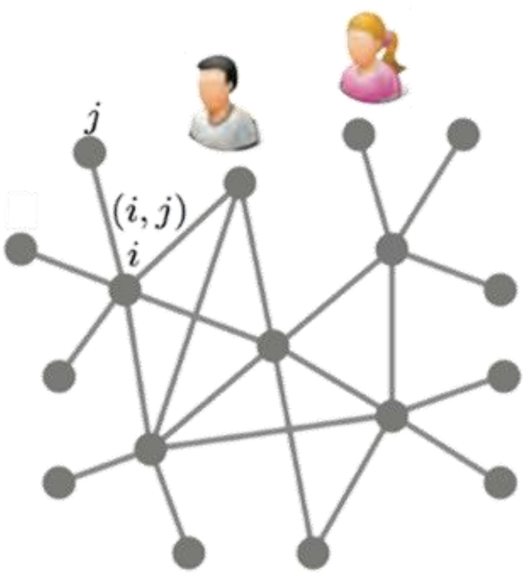


Limitations of Spectral Graph CNNs

- 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

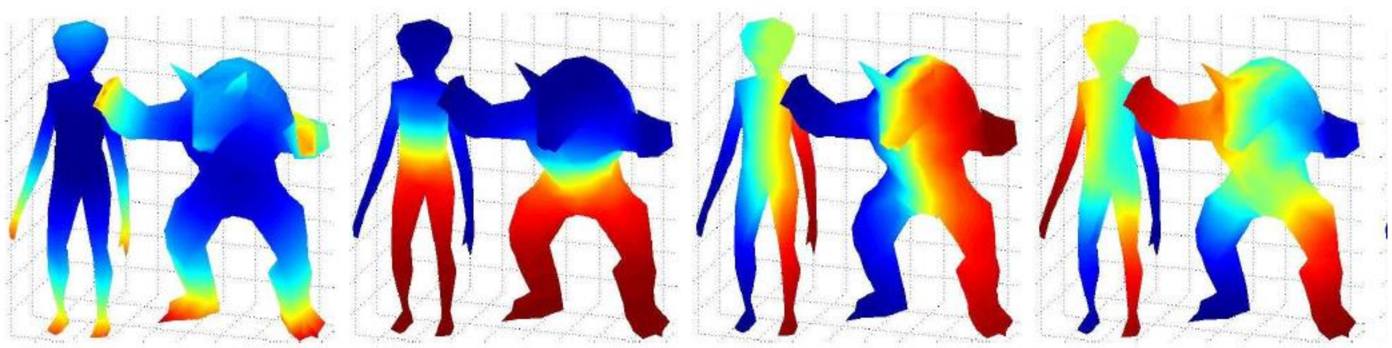
- Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- Vertices $\mathcal{V} = \{1, \ldots, n\}$
- Edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$
- Vertex weights $b_i > 0$ for $i \in \mathcal{V}$
- Edge weights $a_{ij} \geq 0$ for $(i, j) \in \mathcal{E}$
- Vertex fields $L^2(\mathcal{V})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

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



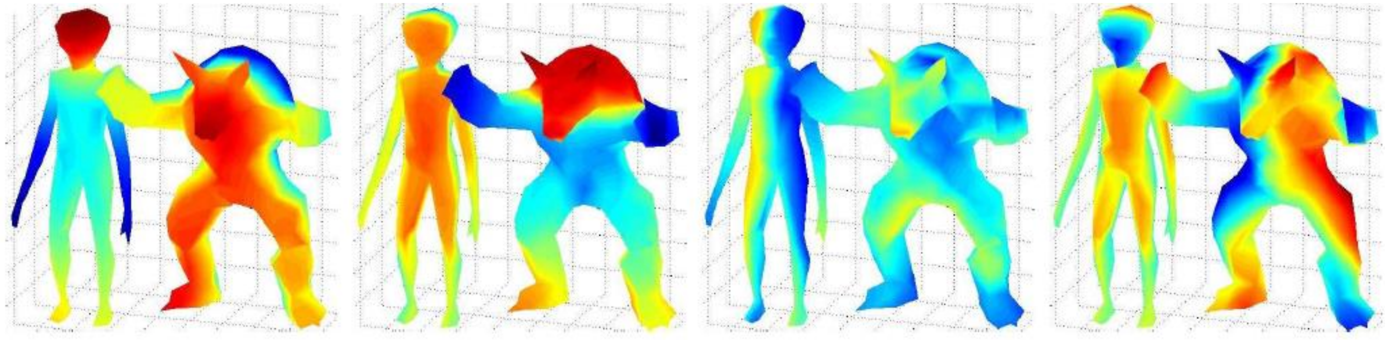
Laplace-Beltrami Spectrum

- 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…..



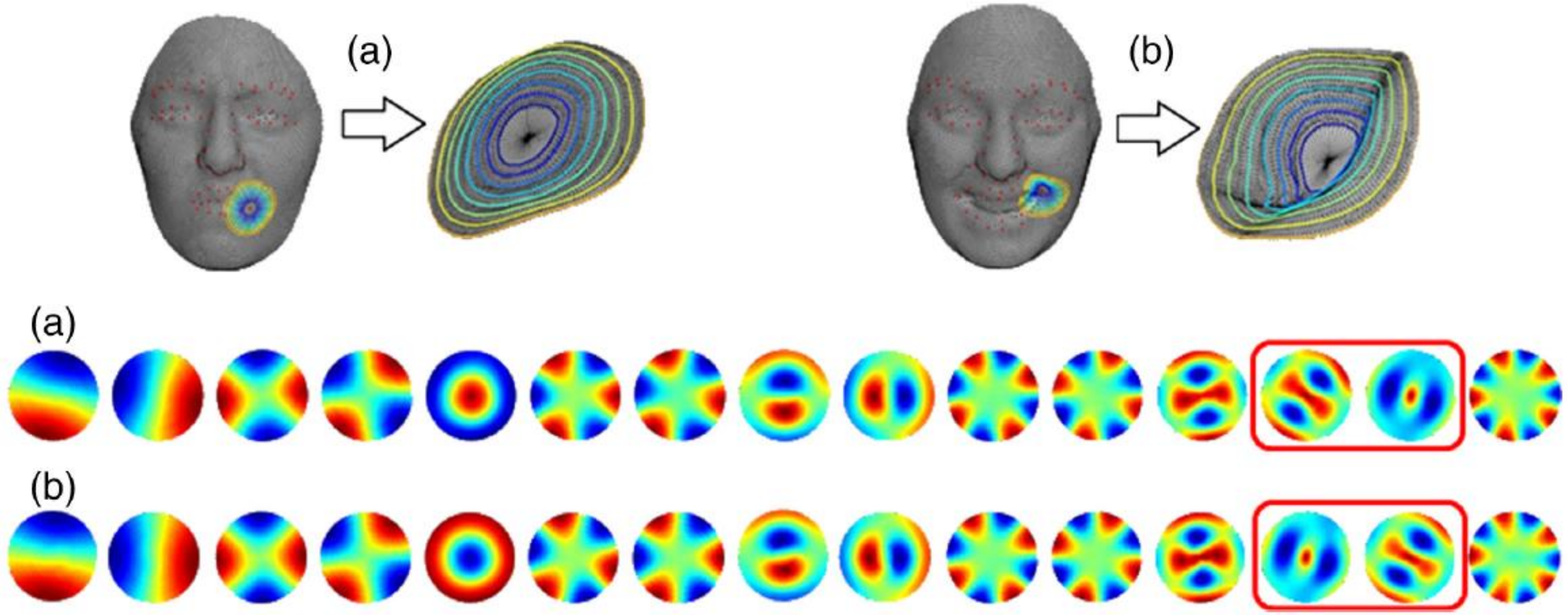
Laplace-Beltrami Spectrum

- 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



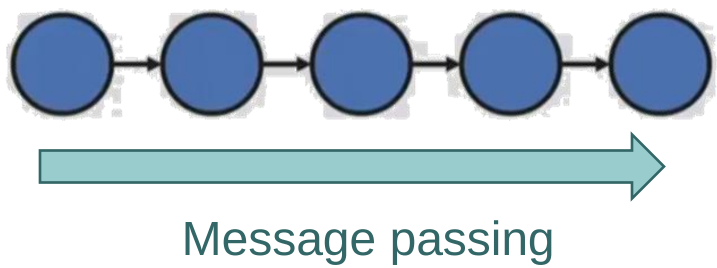
Laplace-Beltrami Spectrum

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

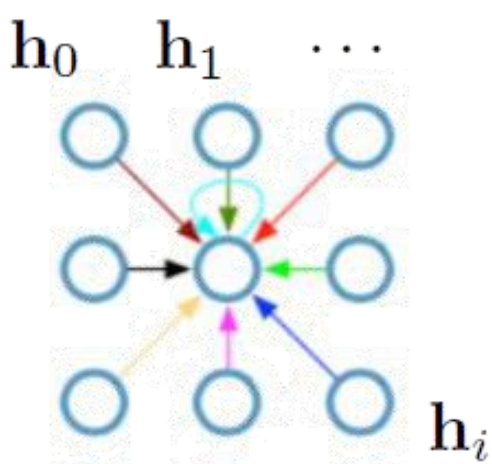


RNN vs CNN motivation for Graph Neural Networks

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



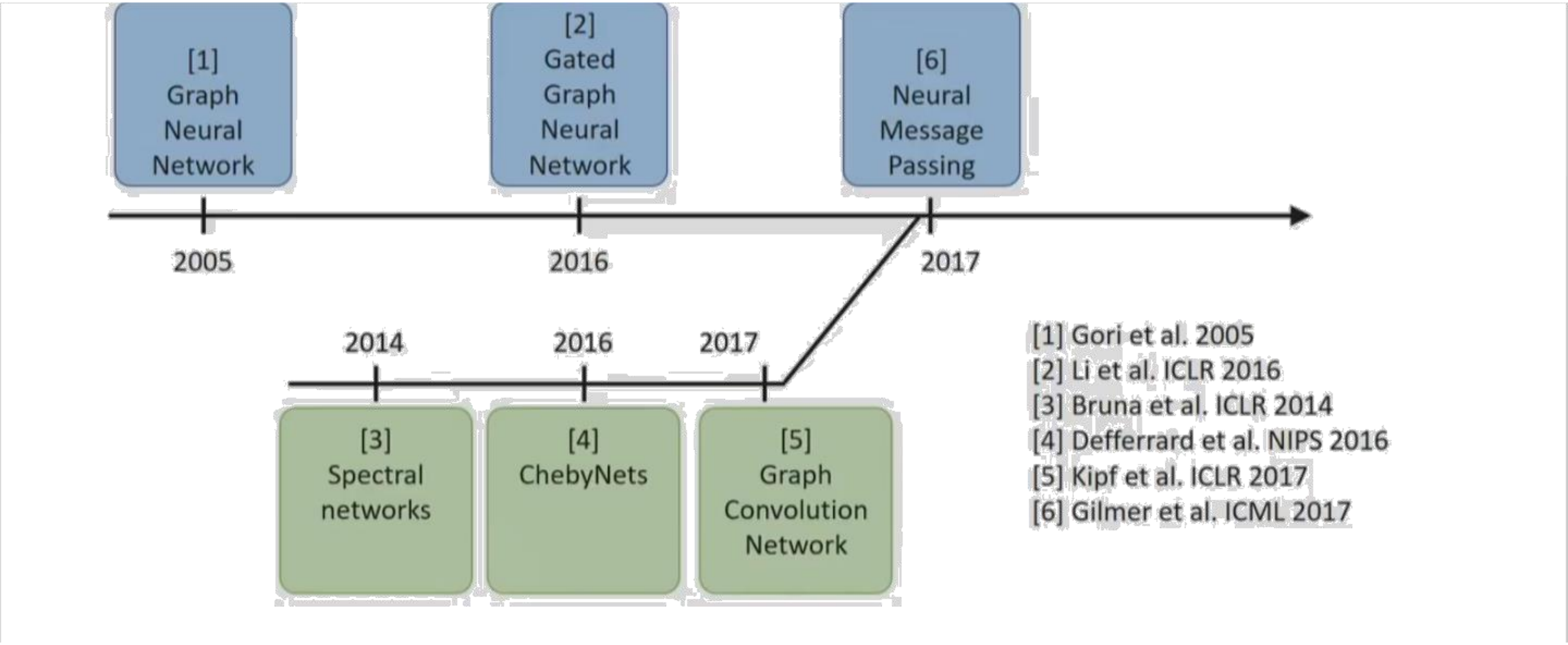
  - Graph Convolutional Nets can be seen equivalently



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

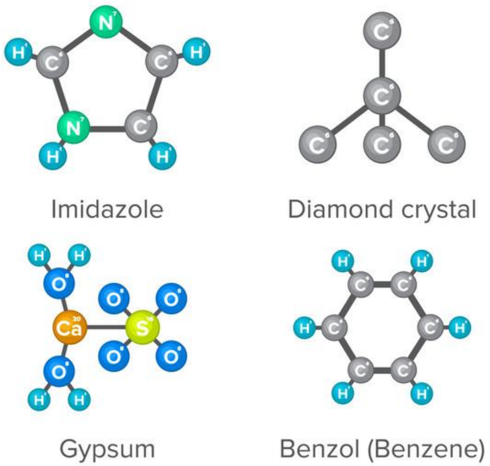
RNN vs CNN motivation for Graph Neural Networks

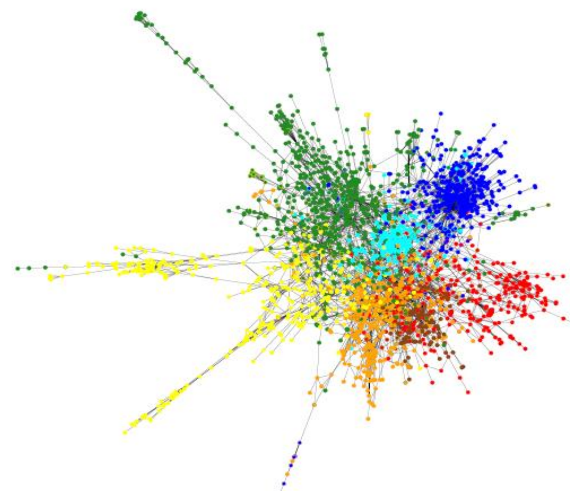
- Both approaches developed from independent paths
  - They were shown to converge to similar formulations



Graph Neural Networks

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





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*