A Network Tour of Data Science

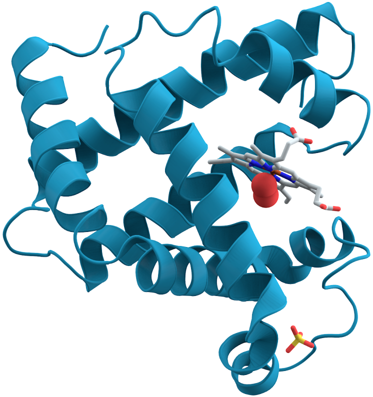
EPFL, 2018-12-11

Learning on Graphs

Michaël Defferrard

École Polytechnique Fédérale de Lausanne

Motivation

$x =$ 

$$y = f(x) = \begin{cases} \text{"toxic"} \\ \text{"non-toxic"} \end{cases}$$

$$y = f(x) = 80\% \text{ toxic}$$

Goal: learn the **unknown** function f, using both **structure** and **features**.

Graph

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

vertex set $\mathcal{V} = \{v_i\}$ with $|\mathcal{V}|$ vertices

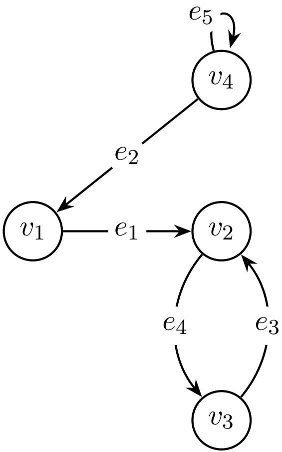
edge set $\mathcal{E} = \{e_i\}$ with $|\mathcal{E}|$ edges

$e_k = (v_i, v_j)$ is an oriented edge from v_i to v_j

adjacency $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$

$A(i,j)$ is the weight of the edge (v_i, v_j)
$A(i,j) = 1$ if edges are not weighted
$A(i,j) = 0$ if $(v_i, v_j) \notin \mathcal{E}$

Example



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

$$\mathcal{V} = \{v_1, v_2, v_3, v_4\}$$

$$\mathcal{E} = \{\underbrace{(v_1, v_2)}_{e_1}, \underbrace{(v_4, v_1)}_{e_2}, \underbrace{(v_3, v_2)}_{e_3}, \underbrace{(v_2, v_3)}_{e_4}, \underbrace{(v_4, v_4)}_{e_5}\}$$

$$A = \begin{pmatrix} 0 & w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & w & 0 & 0 \\ w & 0 & 0 & w \end{pmatrix}, \text{ with edge weight } w$$

Degree

outdegree $D^{out} = \mathrm{diag}(A1) = \mathrm{diag}(d^{out})$

$d^{out}(i) = \sum_j A(i,j)$ is the (weighted) number of edges leaving v_i

indegree $D^{in} = \mathrm{diag}(1A) = \mathrm{diag}(d^{in})$

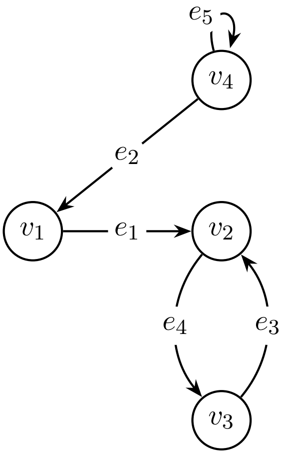
$d^{in}(j) = \sum_i A(i,j)$ is the (weighted) number of edges arriving at v_i

degree $D = \frac{1}{2}(D^{out} + D^{in}) = \mathrm{diag}(d)$

$d(i)$ is the (weighted) number of edges connected to v_i

($D = D^{out} = D^{in}$ for undirected graphs)

Example



$$A = \begin{pmatrix} 0 & w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & w & 0 & 0 \\ w & 0 & 0 & w \end{pmatrix}$$

$$d^{out} = A1 = (w, w, w, 2w)^\top$$

$$d^{in} = 1A = (w, 2w, w, w)$$

$$D = \frac{1}{2}(D^{out} + D^{in}) = \begin{pmatrix} w & 0 & 0 & 0 \\ 0 & \frac{3}{2}w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & 0 & 0 & \frac{3}{2}w \end{pmatrix}$$

Signals

vertex signal a function $x : \mathcal{V} \to \mathbb{R}$ seen as a vector $x \in \mathbb{R}^{|\mathcal{V}|}$

$x(i)$ is the value of x on vertex v_i

edge signal a function $y : \mathcal{E} \to \mathbb{R}$ seen as a vector $x \in \mathbb{R}^{|\mathcal{E}|}$

$y(k)$ is the value of y on edge $e_k = (v_i, v_j)$

Signals are data about vertices and edges, such as features or labels.

Diffusion and random walks

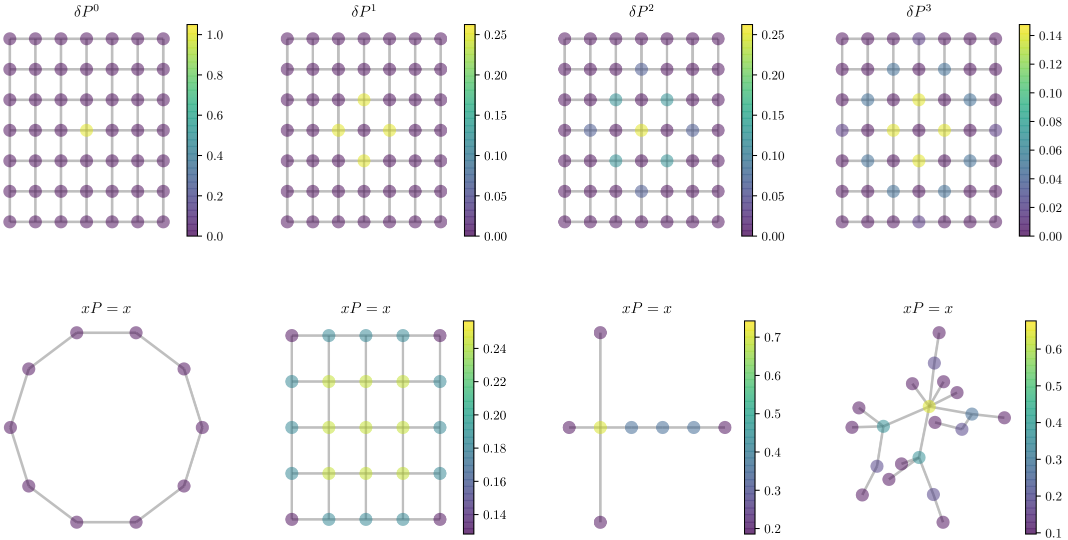
$P = D_{out}^{-1}A$ is the probability transition matrix of a Markov chain

Properties

- P is a right stochastic matrix, i.e., $P\mathbb{1} = \mathbb{1}$
- a random walker starting on v_i has probability $(\delta_i P^k)(j)$ to be on v_j after k steps[1]
- there exists a *stationary* probability vector x such that $xP = x$

[1] The Kronecker delta $\delta_i \in \mathbb{R}^{|\mathcal{V}|}$ has value zero at all vertices but v_i where $\delta_i(i) = 1$.

Example



δP^0
1.0
0.8
0.6
0.4
0.2
0.0
δP^1
0.25
0.20
0.15
0.10
0.05
0.00
δP^2
0.25
0.20
0.15
0.10
0.05
0.00
δP^3
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
$xP = x$
$xP = x$
0.24
0.22
0.20
0.18
0.16
0.14
$xP = x$
0.7
0.6
0.5
0.4
0.3
0.2
$xP = x$
0.6
0.5
0.4
0.3
0.2
0.1

Differential operators

incidence $S(i,k) = \begin{cases} -\sqrt{\frac{A(i,j)}{2}} & \text{if } e_k = (v_i, v_j) \text{ for some } j, \\ +\sqrt{\frac{A(i,j)}{2}} & \text{if } e_k = (v_j, v_i) \text{ for some } j, \\ 0 & \text{otherwise.} \end{cases}$

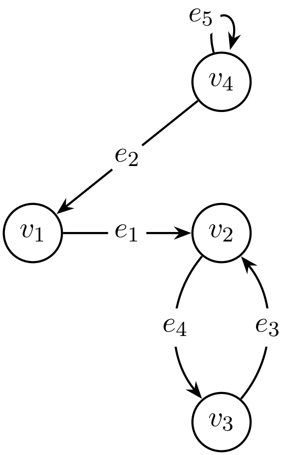
(can leave the $1/\sqrt{2}$ and drop half the edges for undirected graphs)

Laplacian $L = SS^\top = D - \frac{1}{2}\left(A + A^\top\right)$

($A = \frac{1}{2}(A + A^\top)$ for undirected graphs)

Normalized versions: $S_n = D^{-1/2}S$ and $L_n = S_n S_n^\top = D^{-1/2} L D^{-1/2}$

Example



$$A = \begin{pmatrix} 0 & w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & w & 0 & 0 \\ w & 0 & 0 & w \end{pmatrix} \qquad D = \begin{pmatrix} w & 0 & 0 & 0 \\ 0 & \frac{3}{2}w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & 0 & 0 & \frac{3}{2}w \end{pmatrix}$$

$$S = \begin{pmatrix} -\sqrt{w/2} & +\sqrt{w/2} & 0 & 0 & 0 \\ +\sqrt{w/2} & 0 & +\sqrt{w/2} & -\sqrt{w/2} & 0 \\ 0 & 0 & -\sqrt{w/2} & +\sqrt{w/2} & 0 \\ 0 & -\sqrt{w/2} & 0 & 0 & 0 \end{pmatrix}$$

$$L = SS^\top = D - \tfrac{1}{2}(A + A^\top) = \begin{pmatrix} w & -\frac{1}{2}w & 0 & -\frac{1}{2}w \\ -\frac{1}{2}w & \frac{3}{2}w & -w & 0 \\ 0 & -w & w & 0 \\ -\frac{1}{2}w & 0 & 0 & \frac{1}{2}w \end{pmatrix}$$

Differential operators

gradient $\nabla_{\mathcal{G}} x = S^\top x \in \mathbb{R}^{|\mathcal{E}|}$

$(\nabla_{\mathcal{G}} x)(k) = \sqrt{\frac{A(i,j)}{2}}(x(j) - x(i))$, for $e_k = (v_i, v_j)$

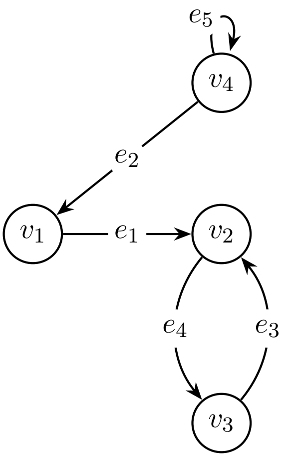
divergence $\operatorname{div}_{\mathcal{G}} y = Sy \in \mathbb{R}^{|\mathcal{V}|}$

$(\operatorname{div}_{\mathcal{G}} y)(i) = \sum_{e_k=(v_j,v_i)} \sqrt{\frac{A(j,i)}{2}} y(k) - \sum_{e_k=(v_i,v_j)} \sqrt{\frac{A(i,j)}{2}} y(k)$

Laplacian $\Delta_{\mathcal{G}} x = \operatorname{div}_{\mathcal{G}} \nabla_{\mathcal{G}} x = Ly \in \mathbb{R}^{|\mathcal{V}|}$

$(\Delta_{\mathcal{G}} x)(i) = d(i)x(i) - \frac{1}{2}\sum_j A(i,j)x(j)$

Example



$$S = \begin{pmatrix} -1 & +1 & 0 & 0 & 0 \\ +1 & 0 & +1 & -1 & 0 \\ 0 & 0 & -1 & +1 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{pmatrix} \qquad \text{(with } w = 2\text{)}$$

$$L = SS^\top = \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 3 & -2 & 0 \\ 0 & -2 & 2 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}$$

let $x = (2, 4, -2, 1)^\top$

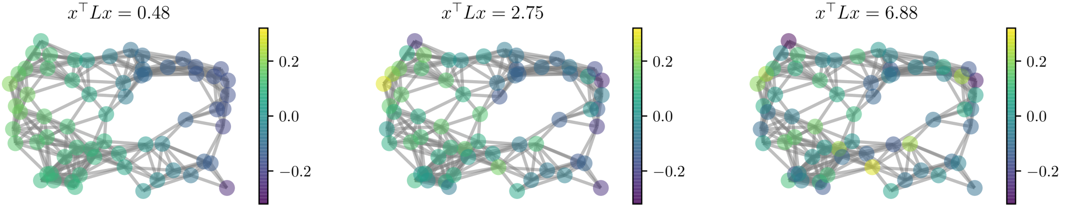
$$y = \nabla_{\mathcal{G}} x = S^\top x = (2, 1, 6, -6, 0)^\top$$

$$z = \operatorname{div}_{\mathcal{G}} y = \Delta_{\mathcal{G}} x = Sy = Lx = (-1, 14, -12, -1)^\top$$

Dirichlet energy

$$x^\top L x = x^\top S S^\top x = \langle S^\top x, S^\top x\rangle = \|S^\top x\|_2^2 = \frac{1}{2}\sum_{i,j} A(i,j)(x(j)-x(i))^2 = \|\nabla_{\mathcal{G}} x\|_2^2$$

This quadratic form is a measure of *smoothness*.



Fourier transform

Introduced to study the heat equation. Why?

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

Joseph Fourier (1768 – 1830)

Fourier basis

Answer: it diagonalizes the Laplace operator.

$$L = U\Lambda U^\top \qquad\qquad u_k = \underset{\substack{u\in\mathbb{R}^{|\mathcal{V}|} \\ \|u\|_2=1 \\ u\perp\{u_1,\ldots,u_{k-1}\}}}{\arg\min} \; u^\top L u$$

eigenvectors $U = (u_1, \ldots, u_{|\mathcal{V}|})$, $U^\top U = I$

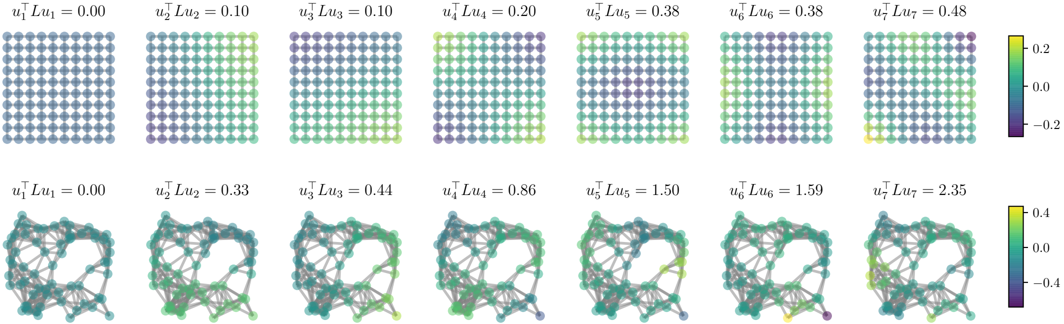
u_k is the k-th Fourier mode s.t. $Lu_k = \lambda_k u_k$

eigenvalues $\Lambda = \mathrm{diag}((\lambda_1, \ldots, \lambda_{|\mathcal{V}|})) = U^\top L U$

$\lambda_k = u_k^\top L u_k$ is the frequency associated to u_k

Example

Fourier mode u_k associated to frequency $\lambda_k = u_k^\top L u_k$.



Fourier transform

transform $\hat{x} = \mathcal{F}_{\mathcal{G}}\{x\} = U^\top x$

$\hat{x}(k) = \langle x, u_k \rangle$ measures how much frequency λ_k is present in x

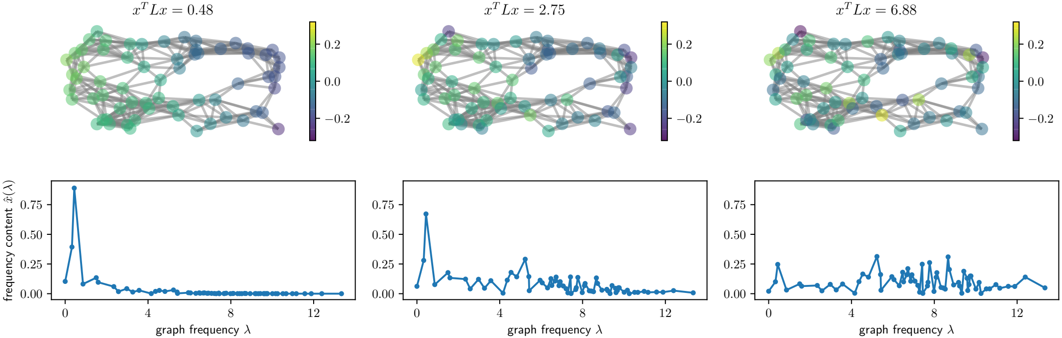
inverse $x = \mathcal{F}_{\mathcal{G}}^{-1}\{\hat{x}\} = U\hat{x} = UU^\top x = Ix$

Interpretation

- change of basis (from vertex to spectral): $x \Rightarrow \hat{x}$ and $L \Rightarrow \Lambda$
- projections of x on the Fourier modes u_k
- harmonic decomposition $x = \sum_k \hat{x}(k) u_k$

Example

Vertex domain representation x and spectral domain representation $\hat{x} = U^\top x$.



Filtering

kernel a function $g : \mathbb{R} \to \mathbb{R}$ that defines the action of the filter

filter an operator acting on signals represented by $g(L)$

A signal $x \in \mathbb{R}^{|\mathcal{V}|}$ is filtered by the kernel g as:

$$y = g(L)x = Ug(\Lambda)U^\top x$$

Step by step

1. take the Fourier transform: $\hat{x} = U^\top x$
2. take an element-wise product with the kernel evaluated at the eigenvalues: $\hat{y} = (g(\lambda_1), \ldots, g(\lambda_{|\mathcal{V}|})) \odot \hat{x}$
3. take the inverse Fourier transform: $y = U\hat{y}$

Functional calculus

What is a **function of a matrix**?

For polynomial functions $g(x) = \sum_k a_k x^k$:

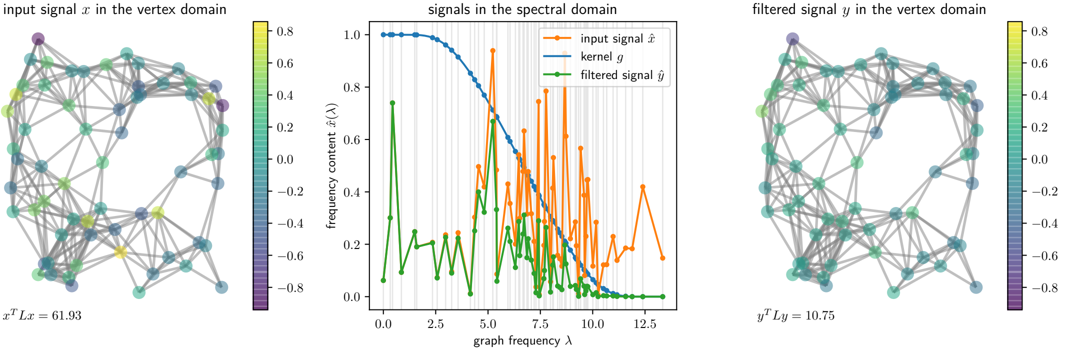
$$g(L) = \sum_{k=0}^{\infty} a_k L^k = U \sum_{k=0}^{\infty} a_k \Lambda^k U^\top = U g(\Lambda) U^\top$$

$$g(\Lambda) = \operatorname{diag}(g(\lambda_1), \ldots, g(\lambda_{|\mathcal{V}|}))$$

Continuous functions through their Taylor expansion:

$$g(L) = e^L = \sum_{k=0}^{\infty} \frac{1}{k!} L^k = U \sum_{k=0}^{\infty} \frac{1}{k!} \Lambda^k U^\top = U g(\Lambda) U^\top$$

Example



Observation: the *low-pass filtered* signal y is much smoother than x!

Convolution without translation?

1D Euclidean convolution:

$(x * g)(i) = \sum_{j=-\infty}^{\infty} x(j)g(i-j) = \langle T_i g, x \rangle$,

where $T_i g$ is a **translation** of the signal g by i steps.

Graph convolution:

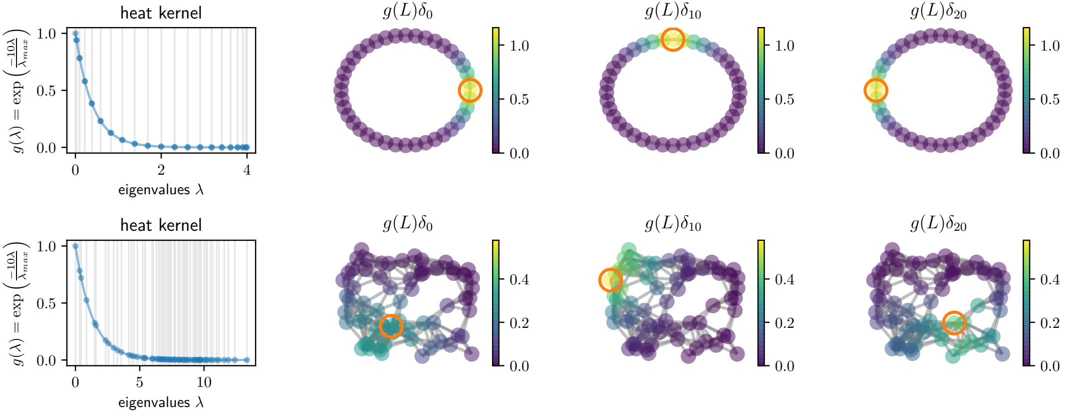
$(x *_{\mathcal{G}} g)(i) = (g(L)x)(i) = \langle \mathcal{T}_i g(L), x \rangle = \langle g(L)\delta_i, x \rangle$,

where $\mathcal{T}_i g$ is the **localization** of the kernel g at node v_i.

We filter x with a kernel g. We cannot convolve x with another signal!

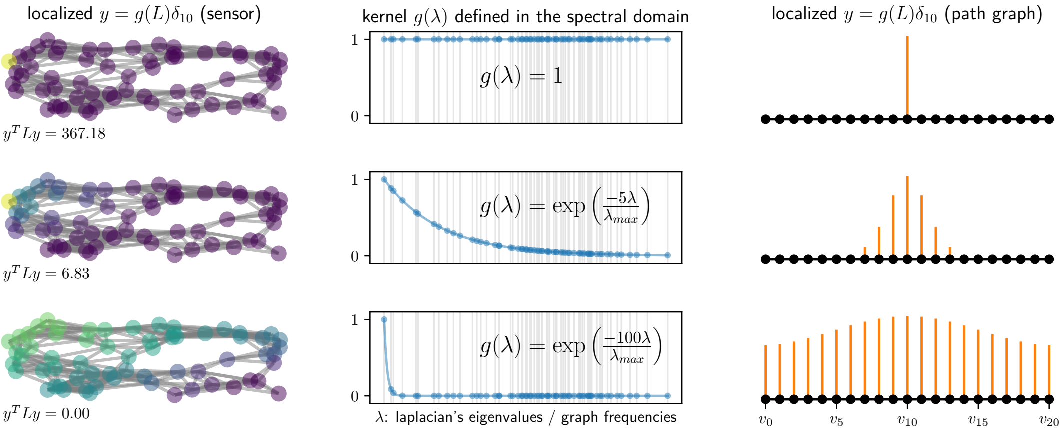
Example: localization vs translation

$$\mathcal{T}_i g(L) = g(L)\delta_i$$



Example: vertex domain kernel visualization

$$\mathcal{T}_i g(L) = g(L)\delta_i$$



Summary so far

1. The adjacency matrix A fully describes a graph $\mathcal{G}$ and acts as a diffusion operator.
2. The incidence matrix S acts as the gradient $S^\top x$ and divergence Sy.
   The Laplacian $L = SS^\top$ is the divergence of the gradient.
3. The Laplacian $L = U\Lambda U^T$ is diagonalized by the Fourier basis U.
4. The Fourier transform $\hat{x} = U^\top x$ shows the frequency content of the signal x.
5. L and Λ (x and $\hat{x}$) are the same operator (function) expressed in different bases.
6. The kernel g filters a signal x as $g(L)x$ with the operator $g(L) = Ug(\Lambda)U^\top$.
7. Kernel $g(\lambda)$ defined in the spectral domain. Localized on v_i as $\mathcal{T}_i g(L) = g(L)\delta_i$.

Filter design

Task: design a kernel $g : \mathbb{R} \to \mathbb{R}$ such that $y = g(L)x$ is the solution of something interesting.

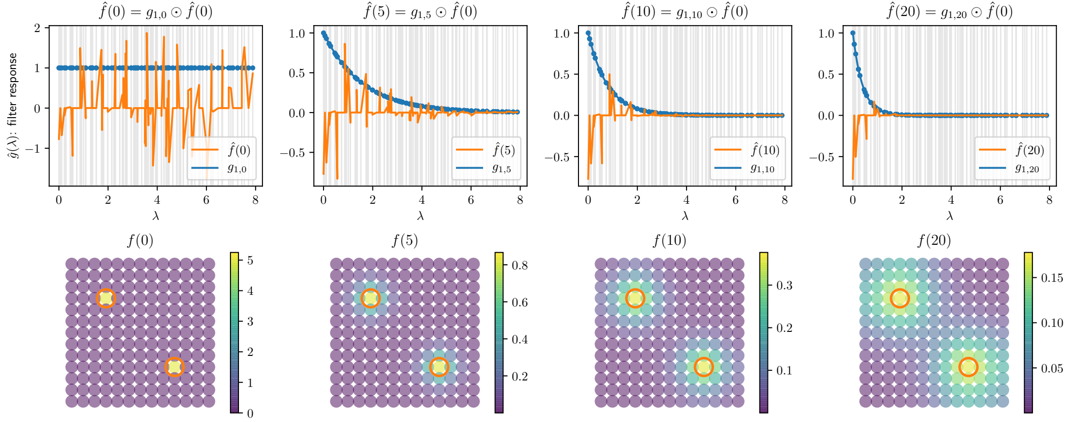
Examples

- Heat diffusion: $g_{\tau t}(\lambda) = \exp(-\tau t \lambda)$
- Wave propagation: $g_{\tau t}(\lambda) = \cos\left(t \arccos\left(1 - \frac{\tau^2}{2}\lambda\right)\right)$
- Projection on a subspace: $g(\lambda) = \begin{cases} 1 & \text{if } \lambda_{min} < \lambda < \lambda_{max}, \\ 0 & \text{otherwise.} \end{cases}$
- Denoising with $\arg\min_y \|y - x\|_2^2 + \tau y^\top L y$: $g(\lambda) = \frac{1}{1+\tau\lambda}$

But what if we don't know the process by which y depends on x, and can't derive g?

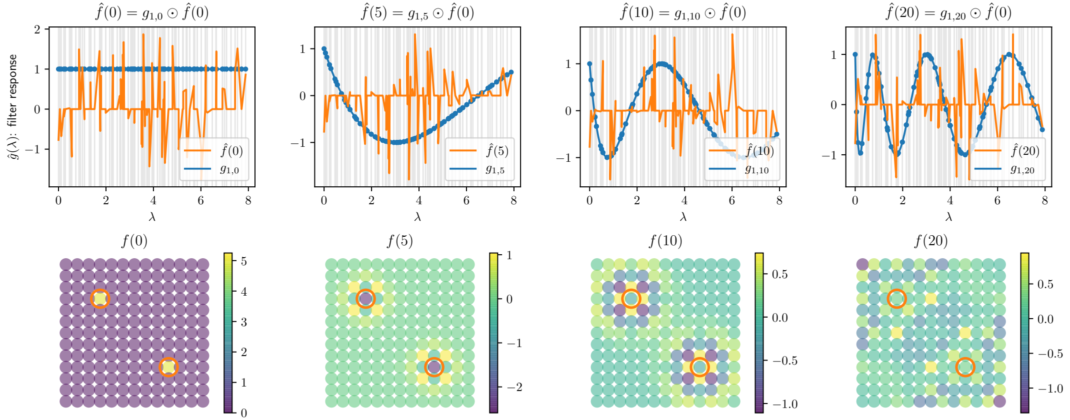
Example: heat diffusion

$$-\tau L f(t) = \partial_t f(t) \quad \Rightarrow \quad f(t) = g_{\tau t}(L) f(0) \text{ with } g_{\tau t}(\lambda) = \exp(-\tau t \lambda)$$



Example: wave propagation

$$-\tau^2 L f(t) = \partial_{tt} f(t) \quad \Rightarrow \quad f(t) = g_{\tau t}(L) f(0) \text{ with } g_{\tau t}(\lambda) = \cos\left(t \arccos\left(1 - \frac{\tau^2}{2}\lambda\right)\right)$$



Learning

Answer: learn the kernel from examples.

Task: approximate the optimal unknown mapping $y = g(L)x$ by a parameterized approximation $y \approx \tilde{y} = g_\theta(L)x$, where θ are the parameters to be learned.

We got:

- a set of examples $\{(x_n, y_n)\}_{n=1}^N$, hopefully large enough
- a cost function to measure how good our approximation is, for example $c(\tilde{y}, y) = \|\tilde{y} - y\|_2^2$

Learning

The goal is to minimize the expected cost $\mathbf{E}_{(x,y)}[c(g_\theta(L)x, y)]$.

The expectation cannot be computed as the distribution $P(x, y)$ is unknown. However, we can compute the empirical risk, an approximation that is the average cost over our training data: $R(g_\theta) = \frac{1}{N}\sum_n c(g_\theta(L)x_n, y_n)$.

Solution: $\hat{\theta} = \arg\min_\theta R(g_\theta)$

Training

How to find $\hat{\theta} = \arg\min_\theta R(g_\theta)$?

A popular optimization algorithm is (stochastic) gradient descent, an iterative algorithm that updates the parameters as

$$\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} c(g_\theta(L)x_i, y_i)$$

upon seeing the example (x_i, y_i).

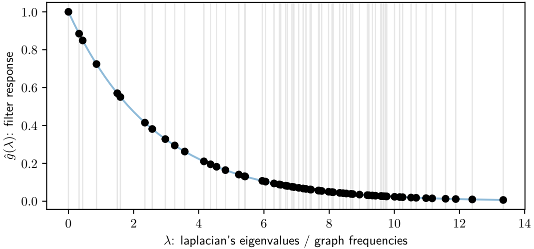
All the computations must be differentiable w.r.t. θ!
In practice, gradients are computed through back-propagation.

Kernel parameterization

Defferrard, Bresson, and Vandergheynst 2016

Non-parametric filter, can learn any filter (n degrees of freedom):

$$g_\theta(\Lambda) = \operatorname{diag}(\theta), \ \theta \in \mathbb{R}^n \ \Rightarrow \ y = U \operatorname{diag}(\theta) U^\top x$$



- Learning complexity is $\mathcal{O}(n)$
- Computational complexity is $\mathcal{O}(n^2)$ (& memory)
- Non-localized in vertex domain

Polynomial parametrization

Defferrard, Bresson, and Vandergheynst 2016

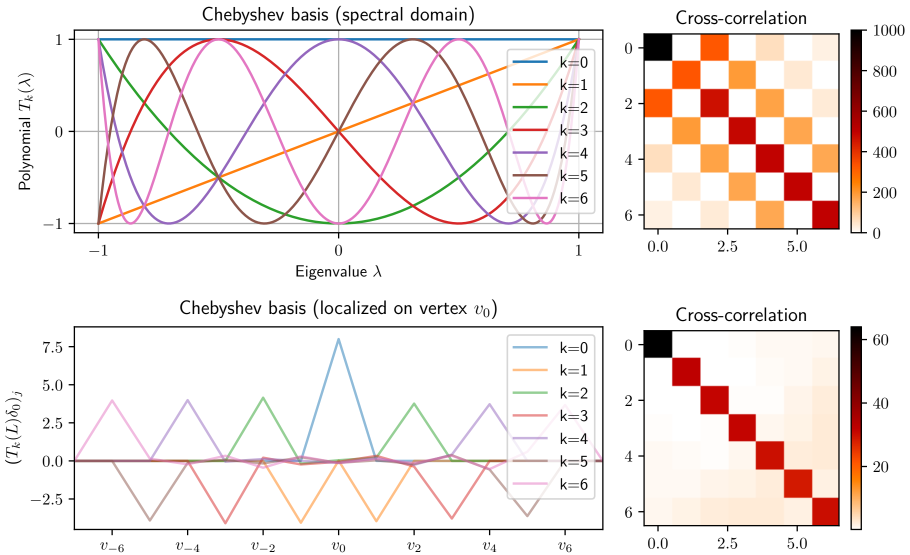
$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k = \sum_{k=0}^{K-1} \tilde{\theta}_k T_k(\tilde{\Lambda}), \quad \tilde{\Lambda} = \frac{2}{\lambda_n}\Lambda - I_n$$

Chebyshev polynomials: $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$
with $T_0 = 1$ and $T_1 = x$

- Can learn any K-localized filter.
- Allows a distributed implementation: only accesses the K-neighborhood.

- K-localized
- Learning complexity is $\mathcal{O}(K)$
- Computational complexity is $\mathcal{O}(K|\mathcal{E}|)$ (same as classical ConvNets!)

Chebyshev polynomials



Fast implementation by recursion

Defferrard, Bresson, and Vandergheynst 2016

$$y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x = \sum_{k=0}^{K-1} \theta_k \bar{x}_k, \quad \tilde{L} = \frac{2}{\lambda_n} L - I_n$$

Recurrence:

$$\begin{aligned} \bar{x}_0 &= x \\ \bar{x}_1 &= \tilde{L}x \\ \bar{x}_k &= T_k(\tilde{L})x = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2} \end{aligned}$$

- Any polynomial can be used. They all have the same representative power. Optimization difficulty might vary.
- Any matrix can be used instead of the Laplacian L, including the adjacency matrix, or even a non-symmetric adjacency or "Laplacian".
- The learned filter parameters θ can be transferred across graphs, i.e., used with different L.

Spatial vs Spectral

Defferrard, Bresson, and Vandergheynst 2016

Convolution on graphs can be **spectrally motivated**.

$$y = U g_\theta(\Lambda) U^\intercal x$$

In the absence of an $O(n \log n)$ Fast Fourier Transform (FFT), which only exists for specific domains, that is however too expensive. $\mathcal{O}(n^3)$ operations for the EVD, plus $\mathcal{O}(n^2)$ operations per forward and backward pass.

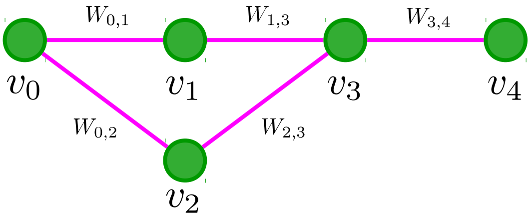
With polynomials, the convolution is however **spatially implemented**.

$$y = g_\theta(L)x = \sum_k \theta_k L^k x = \sum_k \tilde{\theta}_k T_k(\tilde{L})x$$

Leading to many other interpretations: message-passing between nodes, local tangent planes, permutation invariant aggregation, etc.

Weights of paths

$(W^k)_{ij}$ is the sum of all weighted paths of length k between v_i and v_j.

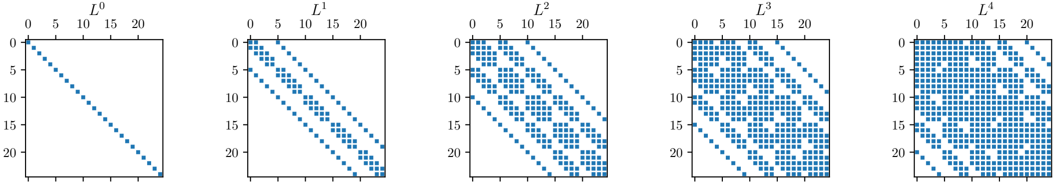


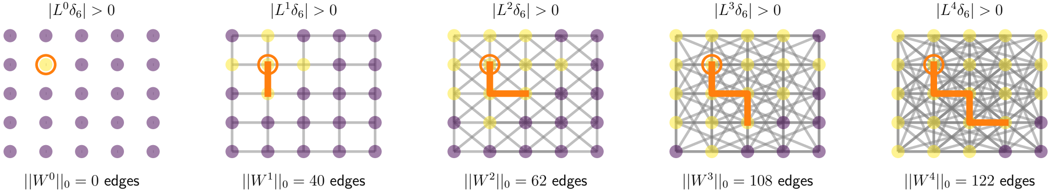
- A path is an ordered set of nodes. Example: (v_2, v_3, v_4).
- $p^k_{ij} = \{(v_i, \ldots, v_j), \ldots, (v_i, \ldots, v_j)\}$ is the set of all paths of length k between v_i and v_j. Example: $p^2_{0,3} = \{(v_0, v_1, v_3), (v_0, v_2, v_3)\}$.
- Path weight $(W^k)_{ij} = \mathrm{weight}(p^k_{ij}) = \sum_{\text{paths}} \prod_{\text{edges } (v_k, v_l)} W_{kl}$.
  Example: $(W^2)_{0,3} = (W_{0,1} \cdot W_{1,3}) + (W_{0,2} \cdot W_{2,3})$.

Neighborhoods

L^k defines the k-neighborhood

Localization: $d_{\mathcal{G}}(v_i, v_j) > K$ implies $(L^K)_{ij} = 0$





Learned combination of neighboring values

$y = \sum_k \theta_k L^k x$ is a linear transformation, where the coefficients are:

- the learned parameter θ_k,
- the k-neighborhood encoded by L^k.

Weighted sum of neighborhoods:

$$y_i = \sum_k \theta_k \bar{x}_k = \underbrace{\theta_0 x}_{\text{own value}} + \underbrace{\theta_1 \bar{x}_1}_{\text{1-neighborhood}} + \underbrace{\theta_2 \bar{x}_2}_{\text{2-neighborhood}} + \cdots + \underbrace{\theta_K \bar{x}_K}_{K\text{-neighborhood}}$$

- Monomials in L: $\bar{x}_k = L^k x$
- Monomials in A: $\bar{x}_k = A^k x$
- Chebyshev polynomials in L: $\bar{x}_0 = x, \bar{x}_1 = \tilde{L}x, \bar{x}_k = T_k(\tilde{L})x = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$

Aggregation function

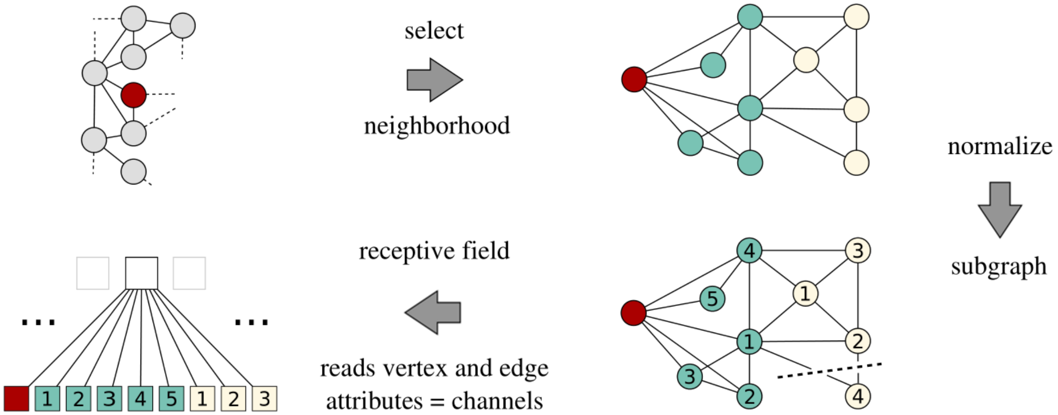
$y = f(x) = \sum_k \bar{x}_k$ is learning how to combine the values $\bar{x}_k$ from the k-neighborhood. The *basic unit* is the neighborhoods, not the nodes.

What else can be done? Any function f that is invariant to the number of neighbors and their permutation.

Goal: map a varying-length representation to a length K representation for $\mathcal{O}(K)$ learning complexity.

Spatial approach: node ordering

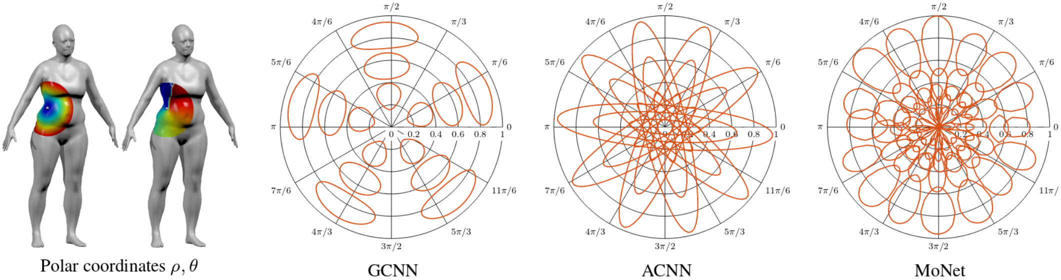
Niepert, Ahmed, and Kutzkov 2016



- anisotropic filters
- require an ordering of the nodes

Spatial approach: patches on the manifold's tangent plane

Monti, Boscaini, Masci, Rodola, Svoboda, and Bronstein 2017



- anisotropic filters
- manifolds only

The need to consider multiple scales

Most data on large graphs exhibit **patterns at multiple scales**.

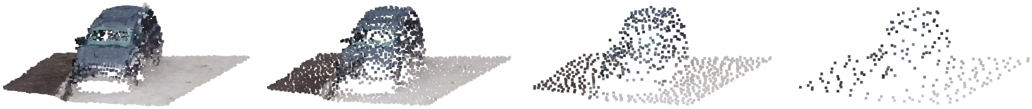
Some filters thus need to have larger receptive fields to capture longer-range dependencies. This can be achieved by:

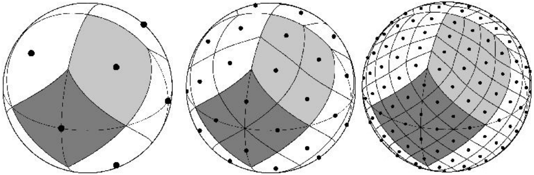
1. increasing the size of the filters (the polynomial order),
2. increasing the number of layers,
3. down-sampling the domain (pooling).

While we can easily do (1) and (2), it can drastically increase the number of parameters to learn. For now, we don't yet have a generic and functional approach to (3).

Coarsening: hierarchical representation

Graph coarsening is certainly an answer to the down-sampling problem.

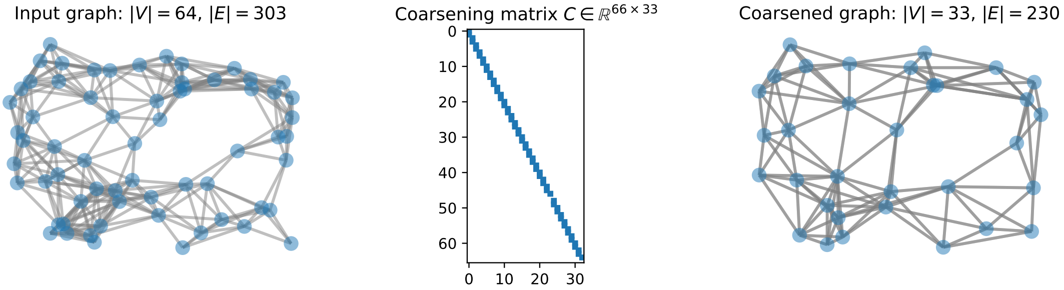




- Easy and well-defined when the domain has a hierarchical structure.

Coarsening: greedy local approach

Defferrard, Bresson, and Vandergheynst 2016

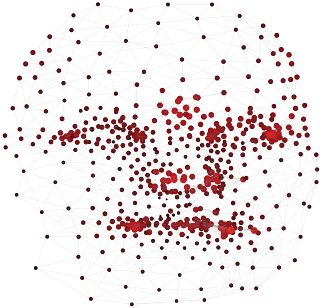


- Greedy node merging (e.g., Graclus, Metis) works well for regular graphs.
- Can be done as pre-processing.
- Conditioned on the structure only.
- Much harder on non-regular graphs.

Learned coarsening: an attention mechanism

Defferrard and Loukas 2018

hard combinatorial problem $\Rightarrow$ learn a **continuous relaxation** of the operation



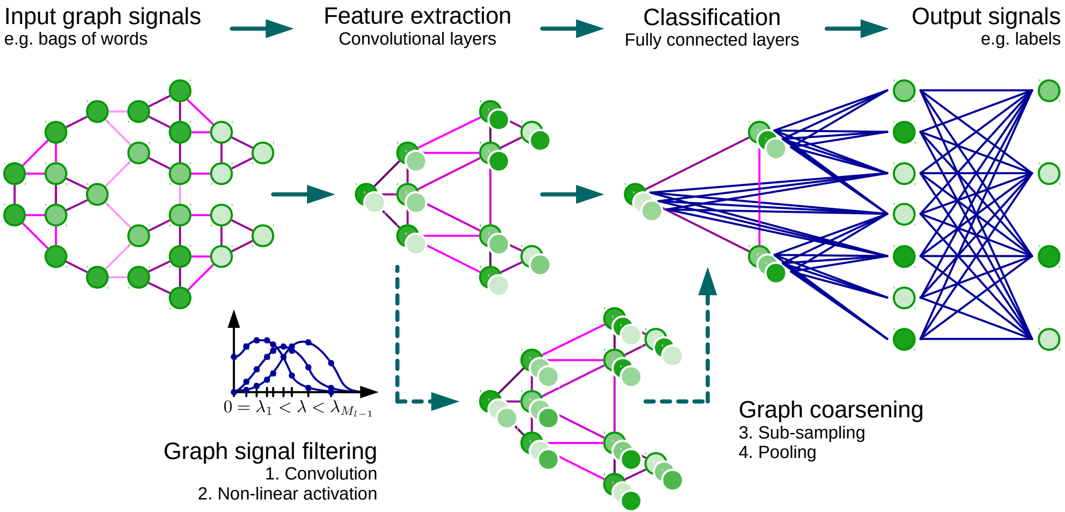
Conditioned on:

1. the structure
2. the features
3. the task

introspection!

Graph ConvNet architecture

Defferrard, Bresson, and Vandergheynst 2016



Multiple kinds of problems: combination of data and tasks

Graphs that model discrete relations

- Social networks
- Graph of citations or hyperlinks
- Molecules (proteins)
- Knowledge graphs

Graphs that represent sampled manifolds

- Meshes (shapes, surfaces)
- Point clouds
- Data on spheres (planets, sky)
- Traffic on roads

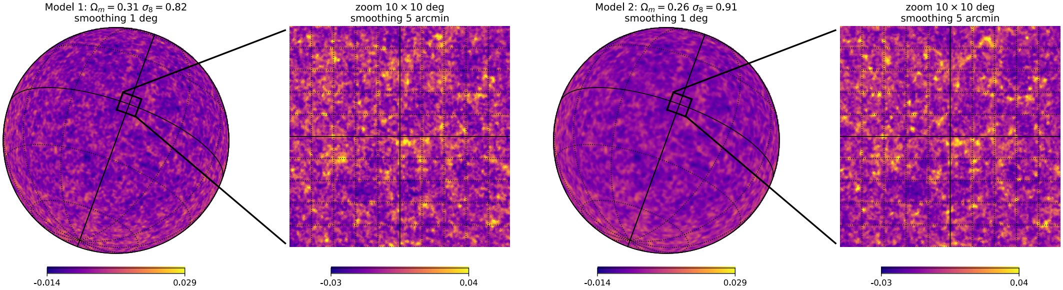
Tasks:

- Node classification or regression (semi-supervized learning)
- Graph classification or regression
- Signal classification or regression

Cosmological application: data & problem

Perraudin, Defferrard, Kacprzak, and Sgier 2018

- Cosmologists devise models of how the universe works.
- We only get to observe one real universe.
- Problem: which simulation is closest to the real thing? A signal classification task.



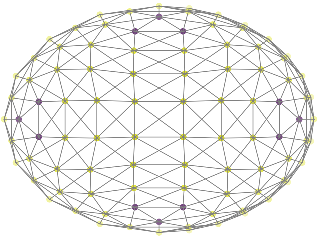
Two mass maps generated from different cosmological parameters.

Cosmology: graph

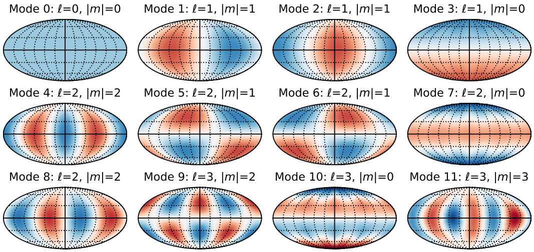
Perraudin, Defferrard, Kacprzak, and Sgier 2018

- Data lives on the sky, a sphere.
- The sphere is discretized, and can be represented by a graph.
- Numerous kind of spherical sky maps in cosmology and astrophysics.

  Cosmic microwave background, galaxy clustering, gravitational lensing.



Sphere discretized by graph.

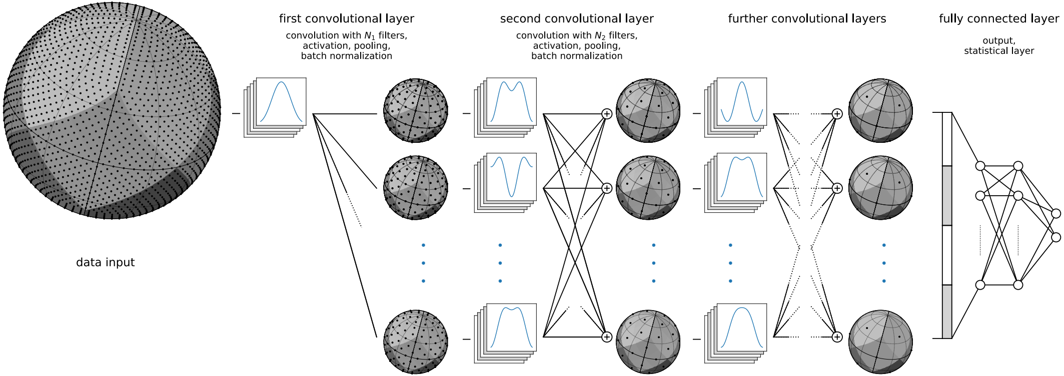


Fourier modes resemble spherical harmonics.

Cosmology: model

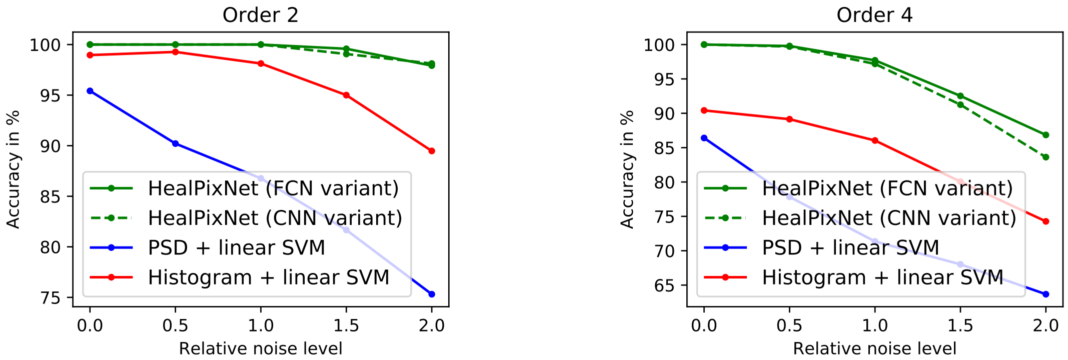
Perraudin, Defferrard, Kacprzak, and Sgier 2018

A classical CNN or FCN architecture, but on the sphere, which is modeled by a graph.



Cosmology: results

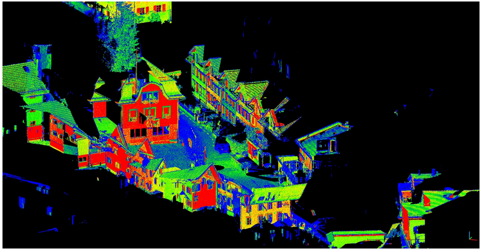
Perraudin, Defferrard, Kacprzak, and Sgier 2018



Significantly better that two standard benchmarks used in cosmology.

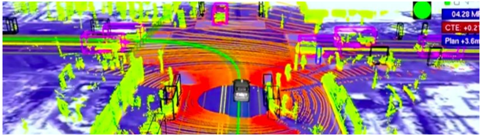
Segmentation of point clouds

remote sensing / surveying



outdoor mapping

indoor mapping



autonomous driving

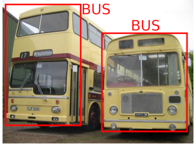
Different classification problems

Goal: assign class labels.

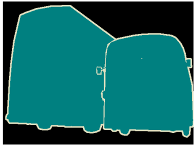
- granularity
- class vs instance

input[1]

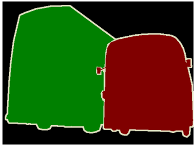
classification



object recognition



semantic seg.



instance seg.

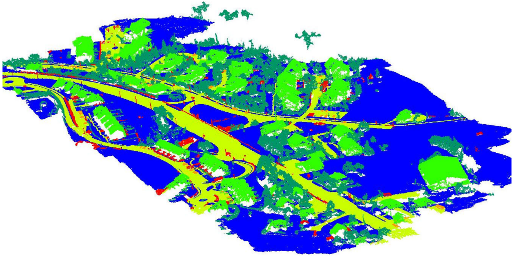
[1] Image source: `https://sthalles.github.io/assets/deep_segmentation_network/object_class_segmentation.png`

Data

input a set of features associated to a set of points

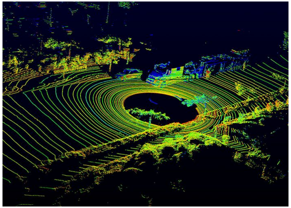
output a label associated to each point

x,y,z coordinates with RGB colors



class labels

Data acquisition



ground LIDAR

aerial LIDAR

aerial images

Our case, aerial images:

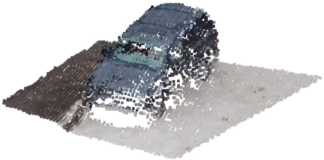
- Drones take aerial pictures of the ground.
- Each point is photographed multiple times from different point-of-views.
- Point cloud constructed by photogrammetry.

Graph

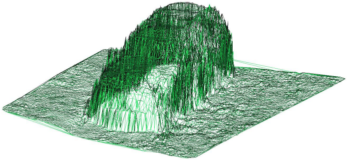
Cherqui, Morsier, and Defferrard 2018

A graph gives:

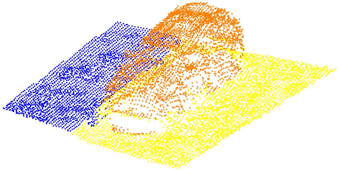
- Neighborhood information, needed for consistent labeling.
- A support, needed for efficient computation.



RGB features



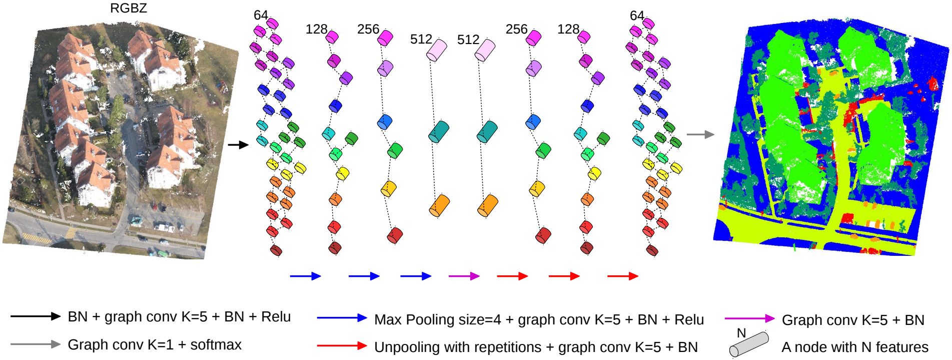
graph



labels

Model

Cherqui, Morsier, and Defferrard 2018



Characteristics:

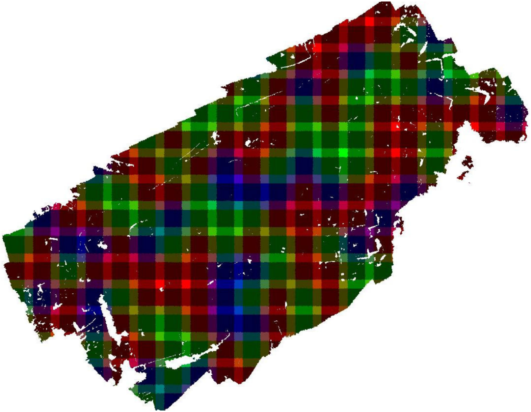
- Dense prediction.
- *Reason* at multiple scales.
- Local decisions.

Main difficulties:

- Large number of points.
- Training samples are of varying sizes.

Data preparation

Cherqui, Morsier, and Defferrard 2018

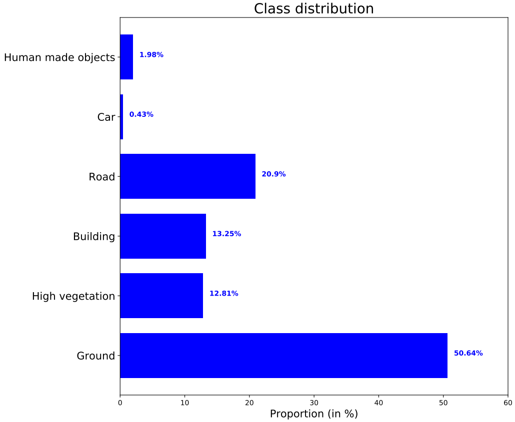


- tiling: $36m \times 36m$ ($48m \times 48m$ with context)
- split: 50% training tiles (green), 16% validation tiles (blue), 35% test tiles (red)

Results with RGBZ

Cherqui, Morsier, and Defferrard 2018

Model	Accuracy Overall (micro)	Accuracy Mean (macro)
Random Forest	75%	53%
Graph ConvNet	86%	68%



Results

Cherqui, Morsier, and Defferrard 2018

Random forest baseline

True label \ Predicted label	Ground	High veg.	Building	Road	Car	Human made obj.
Ground	807919	111645	20379	20821	620	4548
High veg.	99011	134143	6184	2899	200	1159
Building	19336	8995	198616	22489	1716	6586
Road	42386	4866	75775	366655	2342	15912
Car	1891	260	5030	2175	1412	636
Human made obj.	12538	3360	11867	9200	436	3923

Graph ConvNet

True label \ Predicted label	Ground	High veg.	Building	Road	Car	Human-made obj.
Ground	812954	90364	24305	28224	1146	8939
High veg.	24180	210247	3780	2377	15	2997
Building	8065	7025	226223	10405	310	5710
Road	25005	7214	51894	418984	945	3894
Car	220	291	2562	2191	3894	2246
Human-made obj.	6878	9103	8950	2528	1325	12540

Take-home message

Filters can be **designed** to solve known problems.

If the transformation is unknown, **learn** filters from examples.

PS: to practice, try the PyGSP from `https://github.com/epfl-lts2/pygsp`.