Graph Convolutions: More than You Wanted to Know

Presenter: Derrick Blakely

July, 2019

University of Virginia

https://qdata.github.io/deep2Read/

Table of contents

Survey Goals

Review Goals

1. Explain and unify theoretical origins of graph convolutions
2. Explain the key challenges
3. Survey the existing algorithms

1. Theoretical Origins of Graph Convolutions

Theoretical tools underpinning graph convolutions date to 1990s and earlier:

- $<$ 1990: classical convolution, Fourier transform, convolution theorem, uncertainty principle
- Daubechies, 1990: The Wavelet Transform [2]
- LeCun et al., 1995: Convolutional Neural Networks [7]
- Hammond et al., 2011: Wavelets on Graphs via Spectral Graph Theory [4]
- Shuman et al., 2012: The Emerging Field of Signal Processing on Graphs [8]

1. Theoretical Origins of Graph Convolutions

Concept	Continuous	Discrete Euclidean	Graphs
Structure	Time	Regularly structured space	Adjacency matrix
Time to Frequency	Fourier Transform (FT)	Discrete FT (DFT)	Graph FT
Fourier Basis	Complex exponentials	Discrete complex exponentials	Laplacian eigenvectors
Filter Shifting	τ variable	Toeplitz operator	None
Efficiency	FFT	FFT	Polynomial approx. of Fourier basis (and others)
Localization	Continuous Wavelet Transform (CWT)	Discrete Wavelet Transform (DWT)	Spectral graph theory wavelets

How do we fit all of the above in context?

2. Explain the Key Challenges

Graph convolution algos are contingent on the problems they circumvent:

- Generalize Euclidean filtering $\rightarrow$ no regular structure
- No regular structure $\rightarrow$ no vertex domain shift operator
- No vertex domain shift $\rightarrow$ Fourier definition
- Fourier definition $\rightarrow$ need vertex localization
- Need vertex localization $\rightarrow$ use polynomial filter approx.
- No circulant structure $\rightarrow$ can't use traditional Fourier basis
- No traditional Fourier basis $\rightarrow$ Use Laplacian Fourier basis
- Laplacian Fourier basis $\rightarrow$ no FFT
- No FFT $\rightarrow$ need Chebyshev approx.

3. Survey Existing Algorithms

- Spectral Networks (2013) [1]
- Spatially Localized Graph Convolutions (2015) [5]
- Chebyshev Nets (2016) [3]
- GCNs (2016) [6]
- Graph Attention Networks (2017) [9]

Classical Signal Processing Basics

Why does this matter?

At least a few reasons:

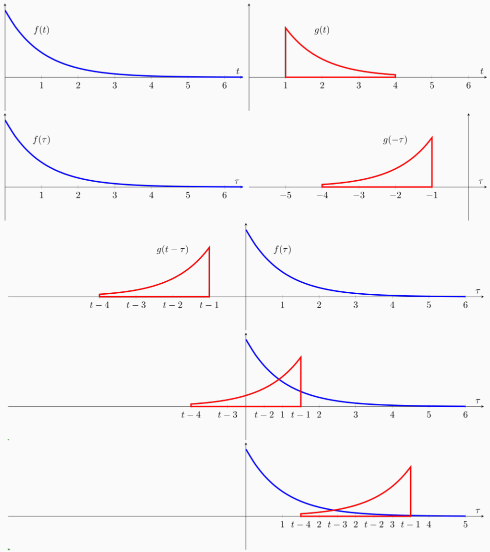
1. Much of the theory and inspiration behind convolutional networks is from classical signal processing; we need to have a hang on some of these concepts
2. Continuous domain (e.g., time) is usually more intuitive than vertex domain
3. We want to understand how our tools have changed as they've been ported to new domains

Classical Convolution

When using a filter g to smooth out a signal f:

$$f(t) = (x * g)(t) = \int_{\mathbb{R}} x(\tau) g(t - \tau) \mathrm{d}\tau \tag{1}$$

Classical Convolution



Convolution Properties

- Continuous case: time-invariance (shift with the τ parameter)
- Discrete case: shift-invariance (shift with a Toeplitz matrix; more on this later)
- Commutative
- Convolution theorem: convolution in time domain = multiplication in Fourier domain (more details coming up)
- Convolution

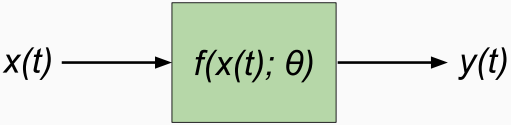
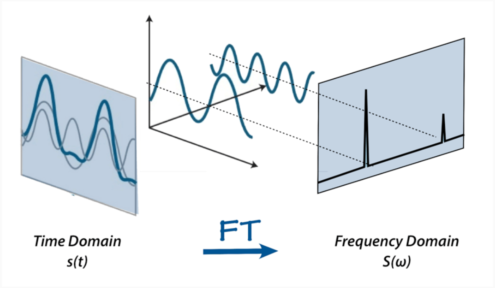


Figure 1: LTI theory: convolution describes linear time-invariant (LTI) systems

The Fourier Transform

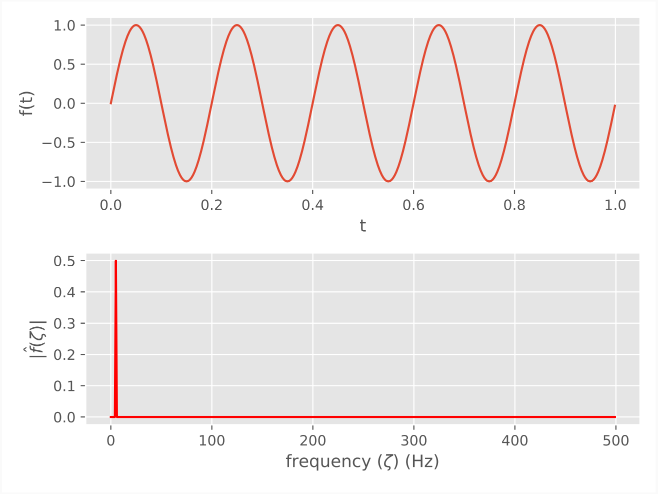
Converts a signal from time domain to frequency domain:

$$\hat{f}(\zeta) = \mathcal{F}\{f(t)\} = \langle f, e^{2\pi i \zeta t} \rangle = \int_{\mathbb{R}} f(t) e^{-2\pi i \zeta t} \mathrm{d}t \tag{2}$$



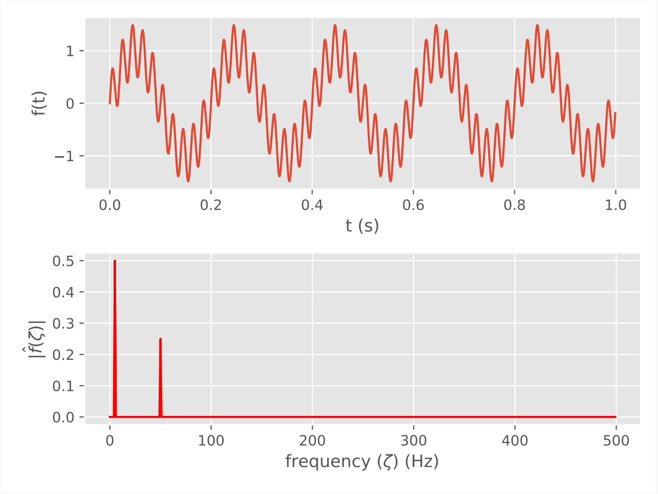
Interpreting Classical Fourier Transforms

Lower values in frequency domain $\rightarrow$ smoother signal



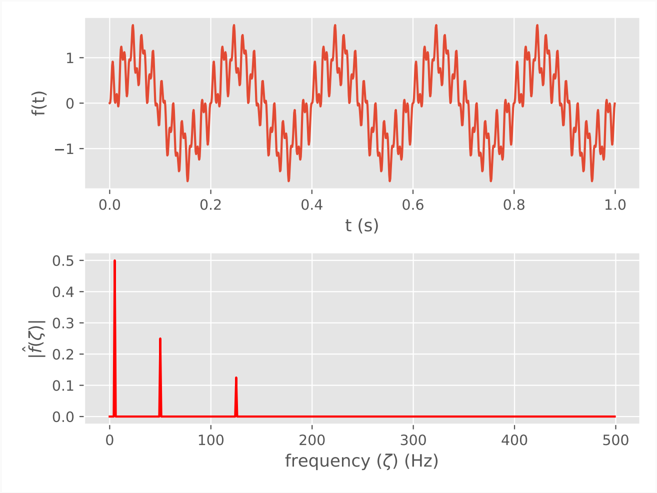
Interpreting Classical Fourier Transforms

Lower values in frequency domain $\rightarrow$ smoother signal



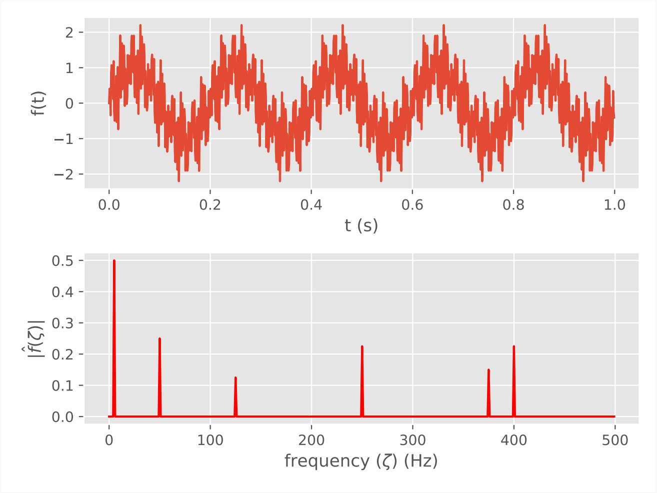
Interpreting Classical Fourier Transforms

Lower values in frequency domain $\rightarrow$ smoother signal



Interpreting Classical Fourier Transforms

Lower values in frequency domain → smoother signal



An Important Point!

In general: a Fourier transform is just an inner product of a signal with some *Fourier basis*:

$$\mathcal{F}\{f\} = \langle f, \Phi \rangle$$

where $\Phi = \{\phi_n\}$ is an orthonormal basis of square-integrable functions. If we define our own Φ, we can define our own Fourier Transform.

Convolution Theorem

Convolution in time domain = point-wise multiplication in frequency domain:

$$\mathcal{F}\{x * g\} = \mathcal{F}\{x(t)\} \cdot \mathcal{F}\{g(t)\} \tag{3}$$

Alternatively:

$$(x * g)(t) = \mathcal{F}^{-1}\{\mathcal{F}\{x(t)\} \cdot \mathcal{F}\{g(t)\}\} \tag{4}$$

Convolution Theorem

Important to us for three reasons:

1. Explains what convolution does in the frequency domain
2. Computing in the Fourier domain can be *faster* than convolving in time domain
3. We'll need it to define graph convolutions later

Convolution Theorem and Frequency Filtering

- Classical filtering: amplifying or attenuating frequencies of the signal

$$\hat{f}_{out}(\zeta) = \hat{f}_{in}(\zeta)\hat{g}(\zeta)$$

  where $\hat{g}(\cdot)$ is the "transfer function."
- Taking an inverse Fourier transform of the above:

$$f_{out}(t) = \int_{\mathbb{R}} \hat{f}_{in}(\zeta)\hat{g}(\zeta)e^{2\pi i\zeta t}\mathrm{d}\zeta = (f_{in} * g)(t)$$

Frequency Filtering

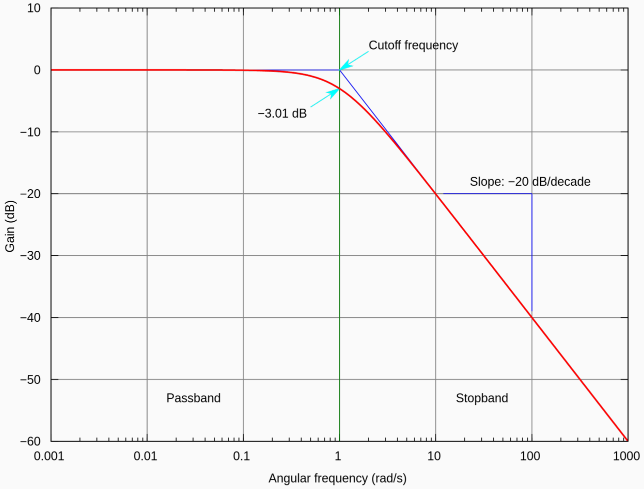


Figure 2: A classical low-pass filter: attenuate the high-frequency components of an input signal (e.g., reduce treble in a song)

The Time-Frequency Localization Trade-off

- As we'll see, constructing *localized* graph filters is non-trivial
- High localization in time domain means low localization in frequency domain and vice versa
- Has deep roots–Heisenberg's uncertainty principle:

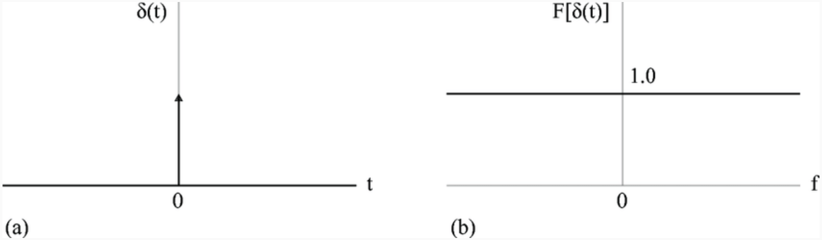
$$\sigma_x \sigma_p \geq \frac{\hbar}{2}$$

- Gabor limit in signal processing:

$$\sigma_t \sigma_f \geq \frac{1}{4\pi}$$

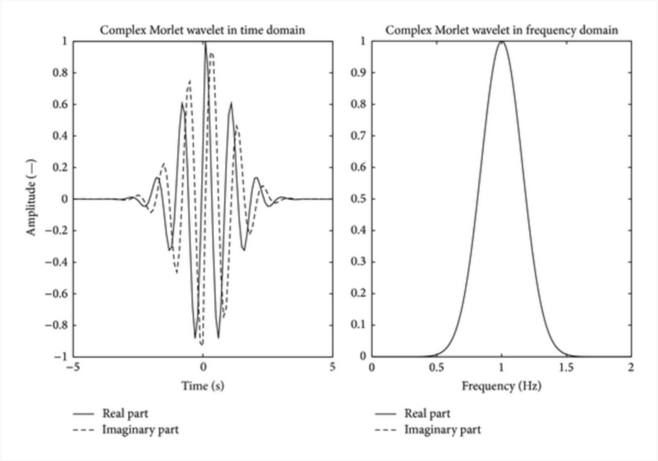
An Extreme Example

A delta function and its Fourier transform:



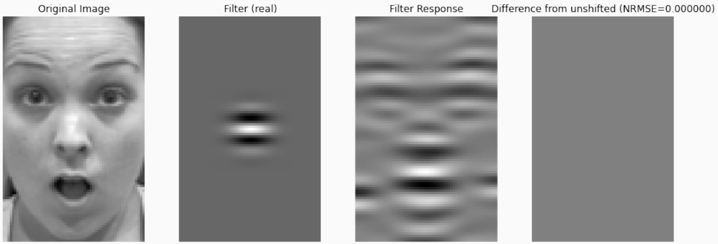
Striking a Balance - Wavelets

Gabor/Morlet wavelets:



Gabor Filters

Time/space and frequency localization makes them very useful for detecting low-level features:



Hypothesis from neuroscience: visual cortex cells are very similar to Gabor filters

Recap

- Convolution uses one function to create a sort of “weighted average” with another function
- Convolution in time domain = multiplication in frequency domain
- A Fourier basis is the eigenbasis of our convolution operator
- Law of nature: hard to localize a filter in both time and frequency domain
- Filters reasonably localized in both are useful for image processing

Discrete Euclidean Convolution

Main Assumptions

Inputs (images, videos, sounds) are composed of patterns that are:

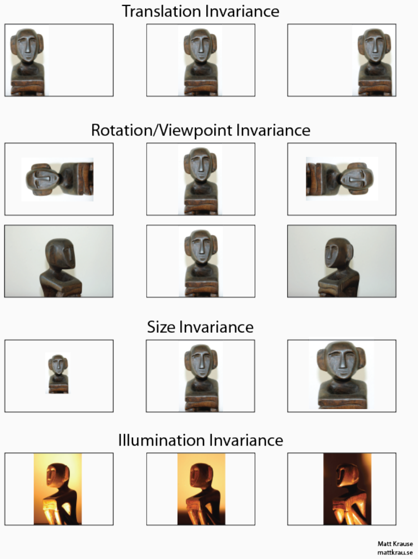
1. Locality
2. Stationarity/translation invariance
3. Hierarchical/multi-scale

Locality

- Nearby pixels are related
- When a CNN applies a filter, the filter is *spatially localized*

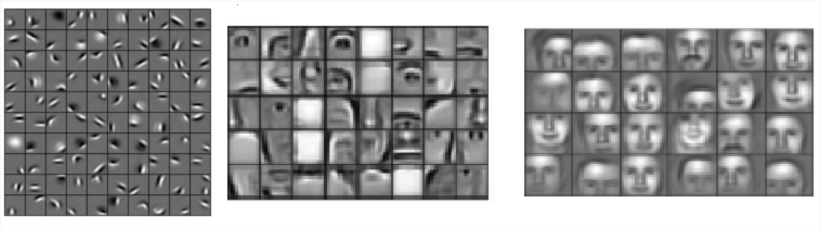
Translation Invariance

Same idea that we saw in LTI theory:



Hierarchical/Multiscale

Result of CNN training: low-level filters resemble Gabor filters, high-level filters look like actual things

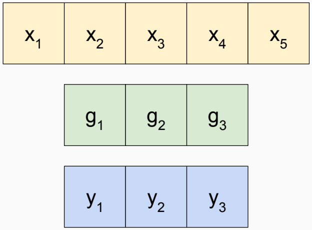


Convolution as a Linear Operator

We can express convolution with a filter as a linear operator:

$$x * g = Gx \tag{5}$$

Example:



$$G = \begin{pmatrix} g_1 & g_2 & g_3 & 0 & 0 \\ 0 & g_1 & g_2 & g_3 & 0 \\ 0 & 0 & g_1 & g_2 & g_3 \end{pmatrix}$$

Convolutions and Circulant Structure

- Circulant matrix: each row is just the previous row shifted to the right (as in previous example)
- Shift-invariant linear system $\iff$ G is a Toeplitz matrix (circulant matrices are Toeplitz matrices)

Convolutions as Linear Operators

G is diagonalized by a Fourier basis:

$$\mathrm{G} = \mathrm{U}\Lambda\mathrm{U}^\top$$

where we can use a discrete Fourier basis as our orthonormal basis:

$$u_k = \left[e^{\frac{i2\pi kn}{N}} | n = 0, 1, ..., N-1\right]^\top$$

Discrete Fourier Transform

The discrete Fourier transform:

$$\hat{x}_k = \sum_{n=0}^{N-1} x_n e^{-\frac{i2\pi}{N}kn} \tag{6}$$

Or using matrices:

$$\hat{\mathsf{x}} = \mathsf{U}^\top \mathsf{x} = \mathcal{F}\{\mathsf{x}\} \tag{7}$$

where U is the “DFT basis” matrix (the columns of U are a set of N independent eigenvectors)

Convolutions as Linear Operators

And so we can compute convolution as:

$$\begin{aligned} \mathrm{x} * \mathrm{g} &= \mathrm{U}\Lambda\mathrm{U}^\top \mathrm{x} \\ &= \mathrm{U} g(\Lambda)\mathrm{U}\mathrm{x} \\ &= \mathrm{U} \begin{bmatrix} \hat{g}_0 & & \\ & \ddots & \\ & & \hat{g}_{N-1} \end{bmatrix} \mathrm{U}\mathrm{x} \\ &= \mathrm{U}\left(\mathrm{U}^\top \mathrm{g} \odot \mathrm{U}^\top \mathrm{x}\right) \\ &= \mathcal{F}^{-1}\{\mathcal{F}\{\mathrm{g}\} \cdot \mathcal{F}\{\mathrm{x}\}\} \end{aligned} \tag{8}$$

Convolutions as Linear Operators

Three important points:

1. If G is circulant, it's a spatially localized filter
2. If G is circulant, $Gx \in \mathcal{O}(n \log n)$ using FFT
3. If G is not circulant, $Gx \in \mathcal{O}(n^2)$

Circulant Matrices and FFT

- Connection to FFT: FFT divide and conquer factorizes G into $\log n$ sub-problems if G is circulant
- Each sub-problem $\sim \mathcal{O}(n) \rightarrow \mathcal{O}(n \log n)$ total (big mood)
- PyTorch, TensorFlow, etc. use FFT-based implementations of conv layers

Graph Convolution

Goal

- Generalize convolution to graphs
- Ensure filter localization
- Ensure efficiency

Challenges

Lots of problems:

- Intuitively: no way to shift a filter matrix around a graph
- No shift-invariance because our filter operator G doesn't have a circulant structure
- How do we define convolution on graphs?
- Can't use discrete Fourier basis. Need to pick a new one
- (As we'll see) Hard to compute efficiently with our new basis

New Fourier Basis: Laplacian Eigenvectors

Laplacian matrix:

$$L = D - A$$

Which acts as a second-derivative operator on graphs:

$$(Lf)(i) = \sum_{j \in \mathcal{N}(i)} W_{i,j} \left[f(i) - f(j) \right]$$

The ∇ Operator

- A pseudo-vector: $\nabla = [\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, ..., \frac{\partial}{\partial x_n}]$

The ∇ Operator

- A pseudo-vector: $\nabla = [\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, ..., \frac{\partial}{\partial x_n}]$
- Gradient: $\nabla f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n}]$

The ∇ Operator

- A pseudo-vector: $\nabla = [\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, ..., \frac{\partial}{\partial x_n}]$
- Gradient: $\nabla f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n}]$
- Divergence: $\nabla \cdot f = \nabla \cdot [f_1, f_2, ..., f_n] = \frac{\partial f_1}{\partial x_1} + \frac{\partial f_2}{\partial x_2} + ... + \frac{\partial f_n}{\partial x_n}$

The ∇ Operator

- A pseudo-vector: $\nabla = [\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, ..., \frac{\partial}{\partial x_n}]$
- Gradient: $\nabla f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n}]$
- Divergence: $\nabla \cdot f = \nabla \cdot [f_1, f_2, ..., f_n] = \frac{\partial f_1}{\partial x_1} + \frac{\partial f_2}{\partial x_2} + ... + \frac{\partial f_n}{\partial x_n}$
- Curl (three dimensions):
  $\nabla \times \mathrm{F} = \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}\right)\mathrm{i} + \left(\frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}\right)\mathrm{j} + \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y}\right)\mathrm{k}$

The ∇ Operator

- A pseudo-vector: $\nabla = [\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, ..., \frac{\partial}{\partial x_n}]$
- Gradient: $\nabla f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n}]$
- Divergence: $\nabla \cdot f = \nabla \cdot [f_1, f_2, ..., f_n] = \frac{\partial f_1}{\partial x_1} + \frac{\partial f_2}{\partial x_2} + ... + \frac{\partial f_n}{\partial x_n}$
- Curl (three dimensions):
  $\nabla \times \mathrm{F} = \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}\right)\mathrm{i} + \left(\frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}\right)\mathrm{j} + \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y}\right)\mathrm{k}$
- Laplacian: $\nabla \cdot \nabla f = \nabla^2 f = \frac{\partial^2 f}{\partial x_1^2} + \frac{\partial^2 f}{\partial x_2^2} + ... + \frac{\partial^2 f}{\partial x_n^2}$

Laplacian for Manifolds

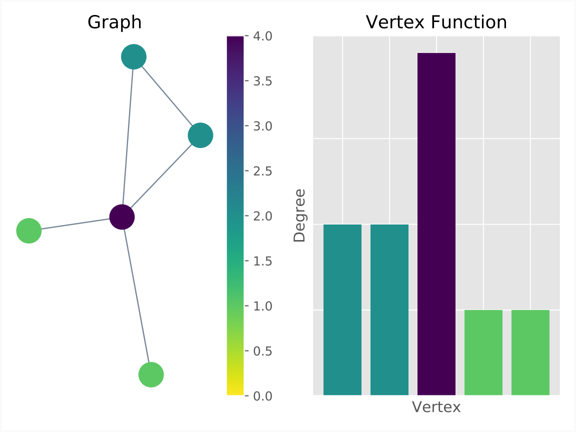
What is the Graph Laplacian?

How do we define $\nabla \cdot \nabla f = \nabla^2 f$ for graphs? Three questions:

1. What does f mean for graphs?
2. What does the gradient ∇f mean for graphs?
3. What does the Laplacian $\nabla \cdot \nabla f$ mean for graphs?

1. Functions Over Graphs

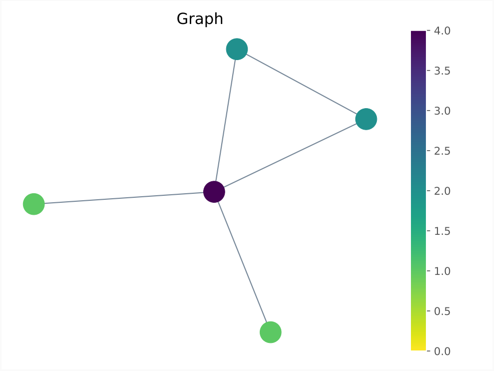
- $f : V \to \mathbb{R}$
- For example: the degree of each node
- In other words: degree of a node is like its potential



2. Gradient of Degree Function

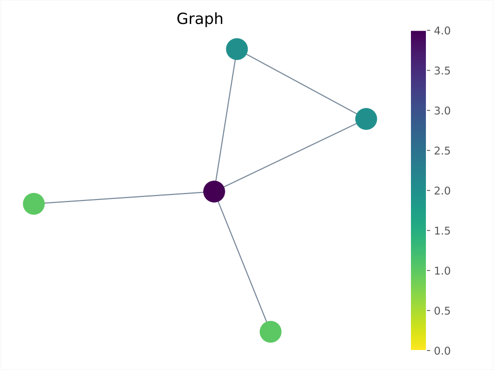
- Need an analog to $\nabla = [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}]$
- Incidence matrix K: each node gets a row and each edge gets a column
- If outgoing edge, $K_{n,e} = -1$
- If incoming edge, $K_{n,e} = 1$
- Neither: $K_{n,e} = 0$

Incidence Matrix K



$$K = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Gradient of Degree Function



$$\nabla f = K^\top f = K^\top \begin{bmatrix} 2 \\ 2 \\ 4 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 2 \\ -3 \\ -3 \end{bmatrix} \tag{10}$$

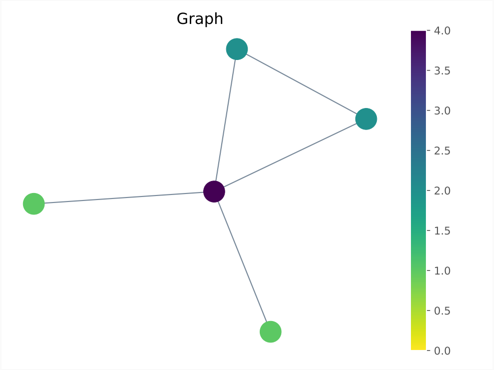
3. Laplacian Operator for Graphs

$$\nabla \cdot \nabla = KK^\top = L = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{bmatrix} \tag{11}$$

$$L = D - A \tag{12}$$

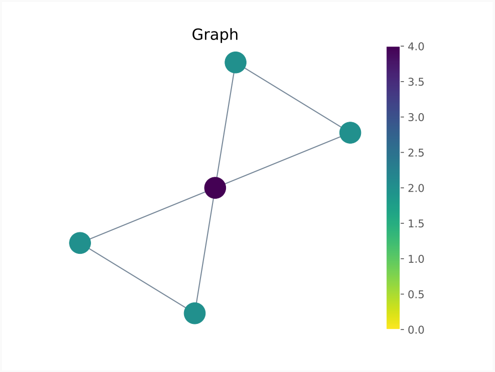
(where D is the degree matrix and A is the adjacency matrix)

Graph Divergence



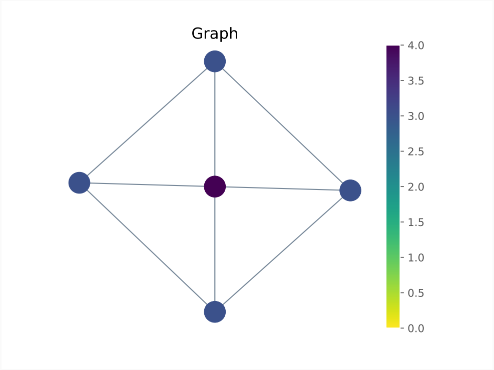
$$\nabla^2 f = KK^\top f = Lf = \begin{bmatrix} -2 \\ -2 \\ 10 \\ -3 \\ -3 \end{bmatrix} \tag{13}$$

Divergence and Smoothness



$$\nabla^2 f = KK^\top f = Lf = \begin{bmatrix} -2 \\ -2 \\ 8 \\ -2 \\ -2 \end{bmatrix} \tag{14}$$

Divergence and Smoothness



$$\nabla^2 f = KK^\top f = Lf = \begin{bmatrix} -1 \\ -1 \\ 4 \\ -1 \\ -1 \end{bmatrix} \tag{15}$$

Summary

Physics	Graphs
Potential: V	Vertex function: f
∇	K: incidence matrix
∇V: field	$K^\top f$: graph gradient
∇^2: Laplacian	$KK^\top$ Graph Laplacian
$\nabla^2 V$: Laplacian of V	$KK^\top f$: Graph Laplacian of f

Fourier Transform and Laplacian

What's the connection to the Laplacian operator?

$$\nabla^2 g = \lambda g \tag{16}$$

for

$$g = e^{2\pi i \zeta t} \tag{17}$$

$$-\nabla^2 e^{2\pi i \zeta t} = -\frac{\partial^2}{\partial t^2} e^{2\pi i \zeta t} = (2\pi\zeta)^2 e^{2\pi i \zeta t} \tag{18}$$

I.e., the Fourier transform = inner product of a function and an eigenfunction of the Laplacian.

Spectral Convolution

With our fancy new Fourier basis and the convolution theorem we can define graph convolution:

$$\begin{aligned}
\mathsf{x} * \mathsf{g} &= \mathcal{F}^{-1}\{\mathcal{F}\{\mathsf{x}\} \cdot \mathcal{F}\{\mathsf{g}\}\} && (19)\\
&= \mathcal{F}^{-1}\{\hat{\mathsf{x}} \cdot \hat{\mathsf{g}}\} && (20)\\
&= \mathcal{F}^{-1}\{\mathsf{U}^\top \mathsf{x} \odot \mathsf{U}^\top \mathsf{g}\} && (21)\\
&= \mathsf{U}\left(\mathsf{U}^\top \mathsf{x} \odot \mathsf{U}^\top \mathsf{g}\right) && (22)\\
&= \mathsf{U} g(\Lambda) \mathsf{U} \mathsf{x} && (23)
\end{aligned}$$

Generic Spectral Filtering

Define our filter as:

$$\hat{g}_\theta(\Lambda) = \mathrm{diag}(\hat{g}_\theta(\lambda_0), ..., \hat{g}_\theta(\lambda_{n-1}))$$

I.e., g is visiting each frequency component λ_l and tweaking it somehow.

$$\begin{aligned} \mathrm{y} = \mathrm{g} * \mathrm{x} &= \mathcal{F}^{-1}\{\hat{\mathrm{g}} \cdot \hat{\mathrm{x}}\} && (24) \\ &= \mathcal{F}^{-1}\{\hat{g}(\Lambda)\mathrm{U}^\top \mathrm{x}\} && (25) \\ &= \mathrm{U}\hat{g}(\Lambda)\mathrm{U}^\top \mathrm{x} && (26) \\ &= \hat{g}(\mathrm{L})\mathrm{x} && (27) \end{aligned}$$

The Localization Problem

- Our generic spectral filter isn't spatially localized
- Hammond et al, 2011: "Wavelets on Graphs via Spectral Graph Theory" [4]
- Defined localized wavelet filters *and* used Chebyshev polynomials to approximate the Laplacian (sound familiar?)

Polynomial Filters

- In general, frequency filtering like this:

$$\hat{f}_{out}(\lambda_l) = \hat{f}_{in}(\lambda_l)\hat{h}(\lambda_l)$$

  is not localized whatsoever in the vertex domain
- A trick: use a polynomial filter like:

$$\hat{h}(\lambda_l) = \sum_{k=0}^{K-1} \theta_k \lambda_l^k$$

Why Polynomial Filters?

Some theoretical and intuitive reasons:

- Tl;dr: polynomial interpolation and Taylor series
- In a local area, think about the frequencies needed to approximate that area
- Basically same idea as Taylor Series, Fourier Series, etc.
- Also: close connection to Lagrange and Chebyshev polynomials (in fact, both of these are used to compute polynomial interpolation in practice)

Localized Graph Filters

Main goal: filter doesn't affect far away nodes. It turns out the Laplacian will help with this:

$$y(i) = \sum_{l=1}^{N-1} \hat{x}(\lambda_l)\hat{g}(\lambda_l)u_l(i) \tag{28}$$

$$= \sum_{j=1}^{N} x(j) \sum_{k=0}^{K-1} \theta_k (\mathsf{L}^k)_{i,j} \tag{29}$$

$$\tag{30}$$

Which is localized because $(\mathsf{L}^k)_{i,j} = 0$ if nodes i and j are more than K hopes away from each other (in fact, this is a graph wavelet)

Localized Filters

Written with matrices:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \tag{31}$$

Which when we apply to x, we get:

$$\mathrm{y} = \mathrm{U} g_\theta(\Lambda) \mathrm{U}^\top \mathrm{x} \tag{32}$$

Problems

Need to compute

$$\mathrm{y} = \mathrm{U} g_\theta(\Lambda) \mathrm{U}^\top \mathrm{x}$$

which is $\mathcal{O}(n^2)$

Chebyshev Nets and GCNs

ChebyNets [3]

Basic idea: we can avoid using the explicit Laplacian Fourier basis by approximating it with Chebyshev polynomials:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\Lambda^k) \tag{33}$$

GCNs

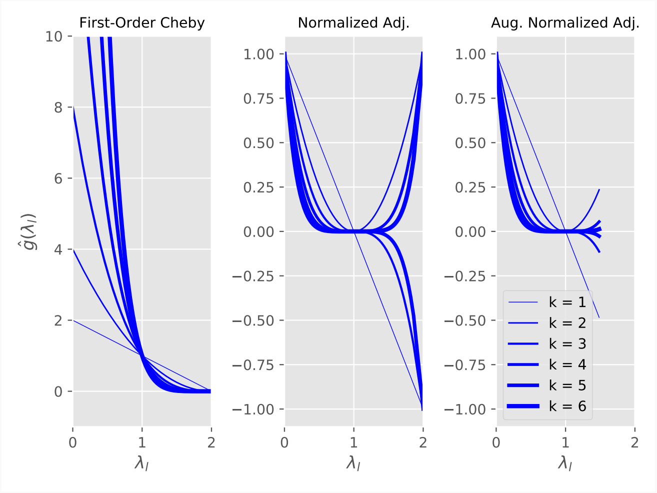
- ChebyNets where we set $k = 1$ to create one layer:

$$g_\theta(\Lambda) = \theta_k T_k(\Lambda) \tag{34}$$

- Use augmented normalized L for numerical stability
- Full layer:

$$\mathrm{Y} = \sigma(\mathrm{AXW}) \tag{35}$$

Spectral Filters



References i

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun.
Spectral networks and locally connected networks on graphs.
arXiv preprint arXiv:1312.6203, 2013.

I. Daubechies.
The wavelet transform, time-frequency localization and signal analysis.
IEEE transactions on information theory, 36(5):961–1005, 1990.

M. Defferrard, X. Bresson, and P. Vandergheynst.
Convolutional neural networks on graphs with fast localized spectral filtering.
In *Advances in neural information processing systems*, pages 3844–3852, 2016.

References ii

D. K. Hammond, P. Vandergheynst, and R. Gribonval.
Wavelets on graphs via spectral graph theory.
Applied and Computational Harmonic Analysis, 30(2):129–150, 2011.

M. Henaff, J. Bruna, and Y. LeCun.
Deep convolutional networks on graph-structured data.
arXiv preprint arXiv:1506.05163, 2015.

T. N. Kipf and M. Welling.
Semi-supervised classification with graph convolutional networks.
arXiv preprint arXiv:1609.02907, 2016.

References iii

Y. LeCun, Y. Bengio, et al.
Convolutional networks for images, speech, and time series.
The handbook of brain theory and neural networks, 3361(10):1995, 1995.

D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst.
The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains.
arXiv preprint arXiv:1211.0053, 2012.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio.
Graph attention networks.
arXiv preprint arXiv:1710.10903, 2017.