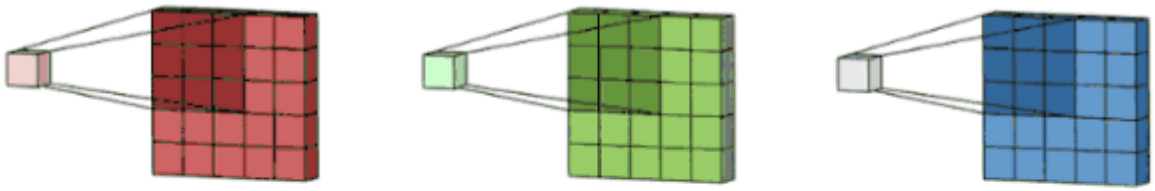
Graph Convolutional Networks for Geometric Deep Learning

Graph Learning and Geometric Deep Learning — Part 2

Flawnson Tong May 14, 2019 · 17 min read

> *Be sure to read an **overview of Geometric Deep Learning** and **the prerequisites** to become familiar with this niche in machine learning.*
>
> *Follow **my Twitter** and join the **Geometric Deep Learning subreddit** for latest updates in the space.*

Graph convolutions are very different from graph embedding methods that were covered in the previous installment. Instead of transforming a graph to a lower dimension, convolutional methods are performed on the input graph itself, with structure and features left unchanged.



Since the graph remains closest to its original form in a higher dimension, the **relational inductive bias** is therefore much stronger.

> ***Inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered.***

There is a type of inductive bias in every machine learning algorithm. In vanilla CNNs for example, the **minimum features** inductive bias states that unless there is good evidence that a feature is useful, it should be deleted. Feature selection algorithms are built upon this assumption. In Geometric Deep Learning, the assumption is relational:

> ***By structuring the data in a way that prioritizes certain patterns, we can improve model performance, even if the data is the same.***

Vanilla convolutions

Generally, a traditional convolutional network consists of 3 main operations:

Kernel/Filter

Think of the kernel like a scanner than "strides" over the entire image. The cluster of pixels that the scanner can scan at a time is defined by the user, as is the number of pixels that it moves to perform the next scan. The kernel aggregates the pixels into a values in grid format to prepare it for pooling.

Pooling

Once again, a "scanner" type of operation is performed, but instead of aggregating a bunch of pixels, pooling singles out only the most important values (max pooling) or averages all the values (mean pooling) and uses that to construct a grid of values for flattening.

Flattening

Flattening simply takes the final grid structure of the pooling operation and truncates them into an array that can then be put through a forward feeding neural model, which

Graph convolutions

Convolutions on graphs however, is much more challenging. Graphs are not so simply manipulated due to their irregular structure. Images are represented on a 2 dimensional Euclidean grid, where a kernel can move left, right, etc. Graphs are non-euclidean, and the notion of directions like up, down, etc. don't have any meaning. **Graphs are more abstract, and variables like node degree, proximity, and neighborhood structure provide far more information about the data.** So the ultimate question is:

How do we generalize convolutions for graph data?

Wet spider silk sort of looks like a graph!

The key to generalizing the convolution is the kernel/filter. We will see that **the biggest difference between Graph Learning approaches is in the kernel,** or what the kernel is operates on.

At a high level, convolutions aggregate information from surrounding or adjacent entities. Convolutions in Deep Learning take this aggregated information to build

Types of Graph Convolutions

There are 2 types of graph convolutions:

> ***Spatial Methods:*** *don't require the use of eigen-stuff*

and

> ***Spectral Methods:*** *requires the use of eigen-stuff*

Both methods are built on different mathematical principles, and it's easy to notice similarities between approaches within each method. However It's probably not very intuitive as to why the use of spectral methods are so popular in graph learning.

This is a high level analysis of some of the most popular architectures.

Spectral Graph Convolutional Networks

It was the research direction of Michaël Defferrard et al that led to the popularization of a new field in both graph theory and signal analytics. This subfield is known as Graph Signal Processing (GSP).

> *GSP is the key to generalizing convolutions, allowing us to build functions that can* ***take into account both the overall structure of the graph and the individual properties of the graph's components.***

GSP uses signal processing functions like the **fourier transform**, which is usually a tool reserved for signals/frequencies, and applies them to graphs. It is the **graph fourier transform** that allows one to introduce the notion of a **"bandwidth" or "smoothness" to a graph.** In the spatial sense of the term, smoothness just means how close the each value of a collection of things are, relative to each other. In the spectral sense, it's a bit more complicated.

vary less within clusters of highly interconnected nodes). In other words, a **clustered graph would be sparse in the frequency domain** allowing for a more efficient representation of the data (frequency, non-euclidean, and spectral domain mean the same thing). For a bit of spectral intuition, you can find a great summary in this great article.

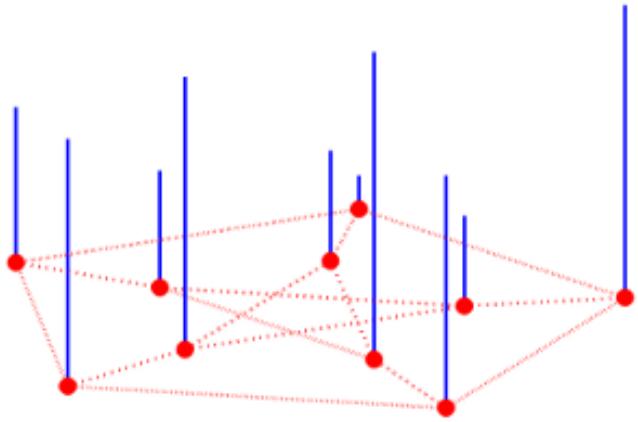


Fig. 1. A random positive graph signal on the vertices of the Petersen graph. The height of each blue bar represents the signal value at the vertex where the bar originates.

A graph with its signals represented in the spatial domain.

In GCNs, **node features and attributes are represented by "signals".** We can then use concepts in signal processing to learn from the data. Usually, a signal isn't just the node or edge feature taken as is, but rather it's **a function that is applied to the feature.**

Convolutions can be computed in by finding the eigendecomposition of the graph Laplacian. Eigendecomposition is a way of factoring a matrix into a set of eigenvectors and eigenvalues. This is also called spectral decompositoin, hence the name Spectral Graph Convolutional Networks. **Calculating the eigenvectors of the Laplacian, returns the Fourier basis for the graph.** Since the calculation is performed on the Laplacian, there will be as many eigenvectors as graph nodes. Directly solving a decomposition is intensive, hence many approaches opt to approximate the spectral

So the overall steps are:

1. Transform the graph into the spectral domain using eigendecomposition
2. Apply eigendecomposition to the specified kernel
3. Multiply the spectral graph and spectral kernel (like vanilla convolutions)
4. Return results in the original spatial domain (analogous to inverse GFT)

The original paper on spectral graph theory was the inspiration for this entire family of graph learning methods. These are some of the most popular.

ChebNets — Defferrard et al

ChebNets is one of the first and most important papers on spectral graph learning. **Spectral convolutions are defined as the multiplication of a signal (node features/attributes) by a kernel**. This is similar to the way convolutions operate on an image, where a pixel value is multiplied by a kernel value.

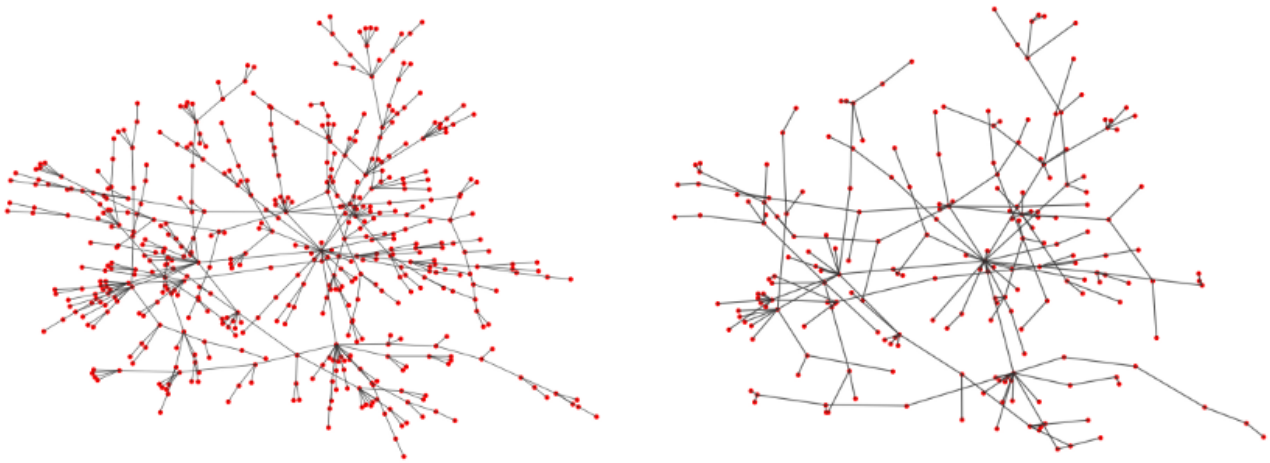
The kernel used in a spectral convolution made of **Chebyshev polynomials of the diagonal matrix of Laplacian eigenvalues.** Chebyshev polynomials are a type of orthogonal polynomials with properties that make them very good at tasks like approximating functions. The kernel is represented by the equation:

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

Where $g\boldsymbol{\theta}$ is a kernel ($\boldsymbol{\theta}$ represents the vector of Chebyshev coefficients) applied to $\boldsymbol{\Lambda}$, the diagonal matrix of Laplacian eigenvalues ($\boldsymbol{\Lambda}$˜ represents the diagonal matrix of scaled Laplacian eigenvalues). $\boldsymbol{k}$ represents the smallest order neighborhood, and $\boldsymbol{K}$ represents the largest order neighborhood. Finally, $\boldsymbol{T}$ stands for the Chebyshev polynomials of the $\boldsymbol{kth}$ order.

> *The kernel equals the sum of all Chebyshev polynomial kernels applied to the diagonal matrix of scaled Laplacian eigenvalues for each order of* **k** *up to* **K-1**.

The original ChebNet paper also introduces **pooling methods**, another key component of the complete vanilla convolution, by using **graph coursening**. This additional step increases efficiency and brings graph convolutions a step closer to their vanilla cousins.



Before and after graph coarsening (Courtesy of Andreas Loukas)

ChebNet implicitly **avoids computing the eigendomposition opting to approximate** it instead. In this regard, ChebNets are very similar to the Graph Convolutional Network by Thomas Kipf et al. GCNs are essentially a first order ChebNet (first order and second order+ are explained in my previous article), with some other simplified operations to reduce complexity.

First order simply means that the metric used to determine the similarity between 2 nodes is based on the node's immediate neighborhood. **Second order** (and beyond) means the metric used to determine similarity considers a node's immediate neighborhood, but also the similarities between the neighborhood structures (with each increasing order, the depth of which nodes are considered increases). The order that is considered within a convolution is denoted by ***K*** (***k*** for the smallest order), in most papers.

limiting a convolution's range to 1st order similarity.

Graph Convolutional Networks (GCNs) — Kipf and Welling

Among the most cited works in graph learning is a paper by Kipf and Welling. The paper introduced spectral convolutions to graph learning, and was dubbed simply as "graph convolutional networks", which is a bit misleading since it is classified as a spectral method and is by no means the origin of all subsequent works in graph learning.

In Kipf and Welling's GCN, a convolution is defined by:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,,$$

eqn. 1

Where $g\boldsymbol{\theta}$ is a kernel ($\boldsymbol{\theta}$ represents the parameters) which is applied (represented by the star) to $\boldsymbol{x}$, a graph signal. $\boldsymbol{K}$ stands for the number of nodes away from the target node to consider (the $\boldsymbol{K}$th order neighborhood, with $\boldsymbol{k}$ being the the closest order neighbor). $\boldsymbol{T}$ denotes the Chebyshev polynomials as applied to $\tilde{\boldsymbol{L}}$ which represents the equation:

$$\tilde{L} = \frac{2}{\lambda_{\max}} L - I_N$$

eqn. 2 (In the original paper, part of the simplification included assuming λ **max = 2**)

Where $\boldsymbol{\lambda\, max}$ denotes the largest eigenvalue of $\boldsymbol{L}$, the normalized graph laplacian. Multiple convolutions can be performed on a graph, and the output is aggregated into $\boldsymbol{Z}$.

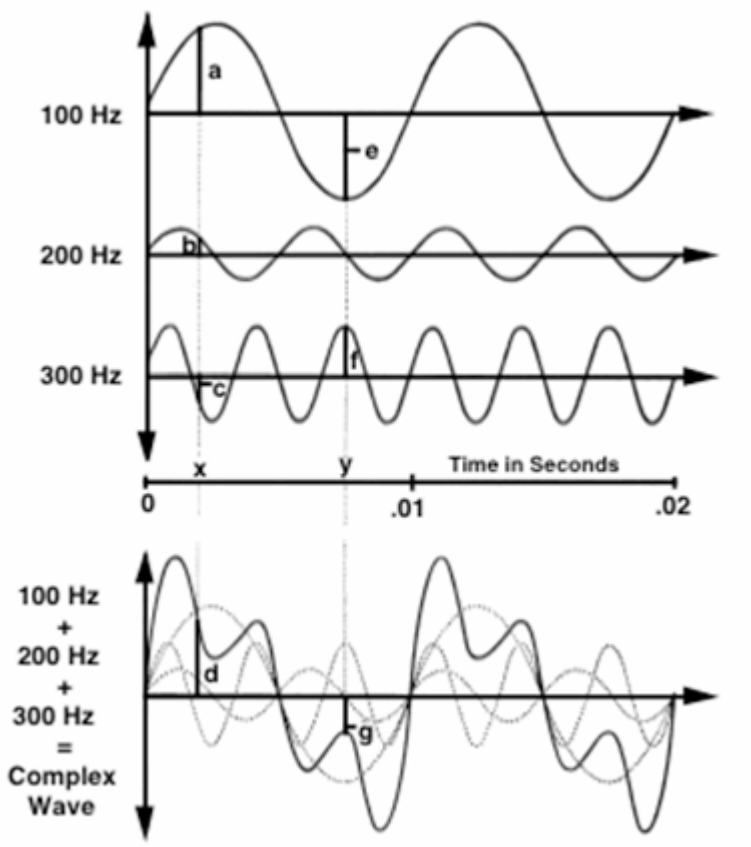
$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \,,$$

eqn. 3

is a matrix of kernel/filter parameters (can be shared over the whole graph), and $\boldsymbol{X}$ is a matrix of node feature vectors. The $\boldsymbol{D}$ to the power of $\boldsymbol{-1/2}$ are parts of a renormalization trick to avoid both exploding or vanishing gradients.

Equation 1 and 3 are the the components that are wrapped in an activation function (Relu, Sigmoid, etc.). The combined result is the layer-wise propagation rule that makes up a single layer of a GCN. The key innovation is the use of the Fourier transform, a popular operation in quantum mechanics (in the study of Q-bits) and signal processing (in the study of waves).

ChebNets and GCNs are very similar, but their largest difference is in their choices for value $\boldsymbol{K}$ in eqn. 1. In a GCN, the layer wise convolution is limited to $\boldsymbol{K = 1}$. This is intended to **alleviate the risk of overfitting on a local neighborhood of a graph.** The original paper by Kipf et al. further explain how this makes the architecture linear with respect to the Laplacian, simplifying the overall model complexity.



Component sound waves coming together to make a single sound (Courtesy of Hearing health matters)

are component decompositions of the latent graph. This is analogous to how component frequencies are decompositions of a sound wave; component frequencies are node features (as signals), and the sound wave signal is a latent graph.

GCNs performed well in node classification tasks and other graph applications, but the main drawback is how **eigenvalues tend to cluster together in a very small range**, with large gaps in between each cluster. The same can be emphasized to a lesser degree for ChebNets. This is problem later solved by CayleyNets.

Fast/Simplified Graph Convolutions (FastGCNs/SGCs)

The biggest drawbacks to Spectral Convolutions is their tendency to be very computationally expensive. The kernel is defined in Fourier space and graph Fourier transforms are notoriously expensive to compute. It requires multiplication of node features with the eigenvector matrix of the graph Laplacian, which is a $O(N^2)$ operation for a graph with N nodes.

And so the problem that IBM researchers set out to solve was how to improve the computational efficiency while still retaining the best possible results. FastGCN was thus born out of the need for scalable spectral graph convolutions. The project essentially modified the original GCN architecture to be able to use **Monte Carlo** (biased random sampling method) approaches to consistently estimate the integrals, which allowed for **batch training,** reducing the overall training time.

Table 4. Test Accuracy (%) on text classification datasets. The numbers are averaged over 10 runs.

Dataset	Model	Test Acc. ↑	Time (seconds) ↓
20NG	GCN	87.9 ± 0.2	1205.1 ± 144.5
	SGC	88.5 ± 0.1	19.06 ± 0.15
R8	GCN	97.0 ± 0.2	129.6 ± 9.9
	SGC	97.2 ± 0.1	1.90 ± 0.03
R52	GCN	93.8 ± 0.2	245.0 ± 13.0
	SGC	94.0 ± 0.2	3.01 ± 0.01
Ohsumed	GCN	68.2 ± 0.4	252.4 ± 14.7

SGC	75.9 ± 0.3	4.00 ± 0.04

Table 5. Test accuracy (%) within 161 miles on semi-supervised user geolocation. The numbers are averaged over 5 runs.

Dataset	Model	Acc.@161↑	Time ↓
GEOTEXT	GCN+H	60.6 ± 0.2	153.0s
	SGC	61.1 ± 0.1	5.6s
TWITTER-US	GCN+H	61.9 ± 0.2	9h 54m
	SGC	62.5 ± 0.1	4h 33m
TWITTER-WORLD	GCN+H	53.6 ± 0.2	2d 05h 17m
	SGC	54.1 ± 0.2	22h 53m

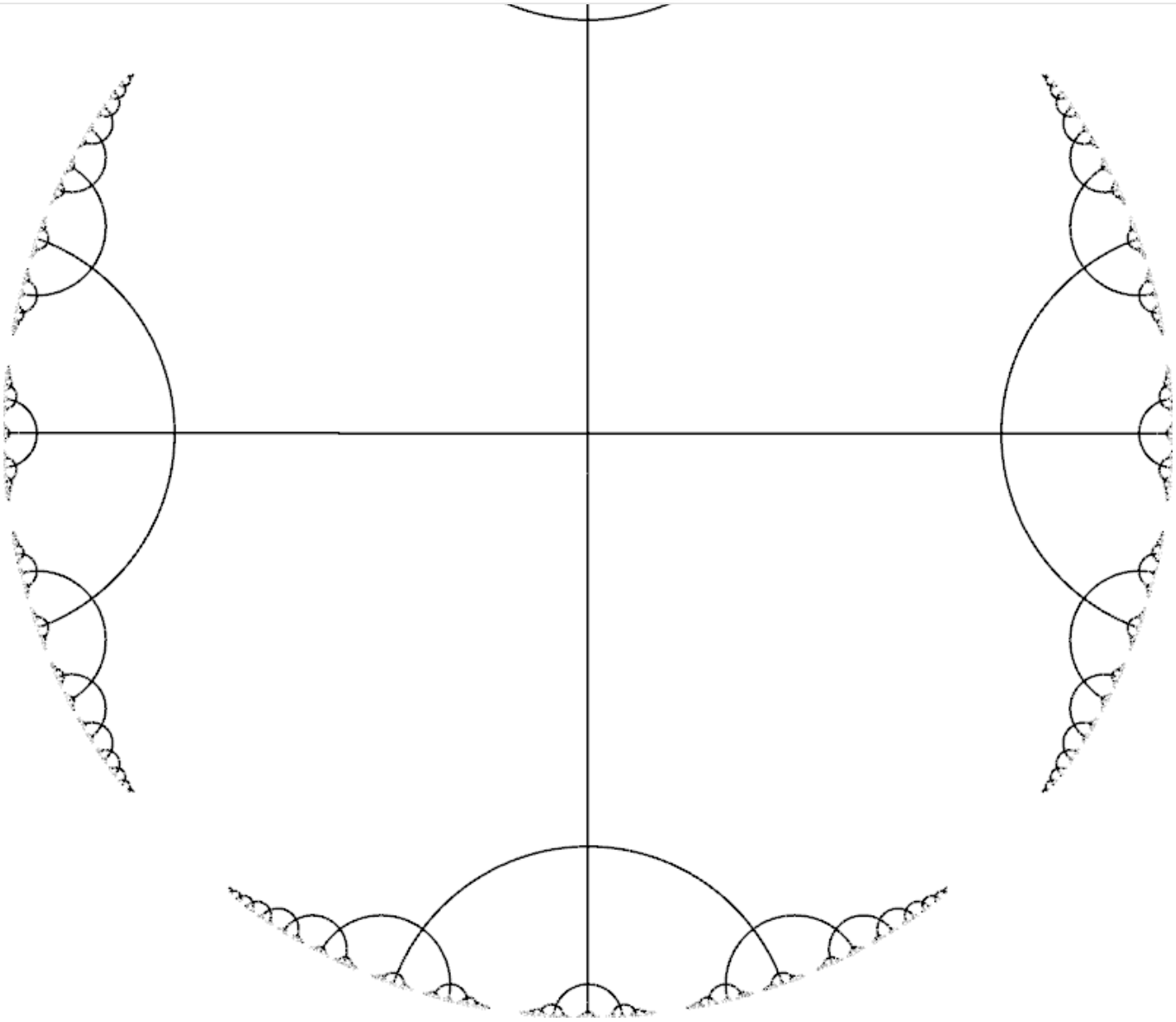
Simplified Graph convolutions (SGCs) on the other hand, approached the problem differently; instead of improving computational efficiency, perhaps reducing computational complexity is the natural solution. Their experiment results are on shown on the diagram to the left. The premise behind their hypothesis:

> " We hypothesize that the nonlinearity between GCN layers is not critical — but that the majority of the benefit arises from the local averaging."

The project **removed the nonlinear transition functions between each convolutional layer.** The final softmax activation allowed for probabilistic outputs that can be optimized with stochastic gradient descent.

CayleyNets

CayleyNets improves upon ChebNet's high computational expense problem using a secret ingredient. As indicated by the name, this approach uses the **Cayley transform** (represented by unit half-circle) to fix the problem.

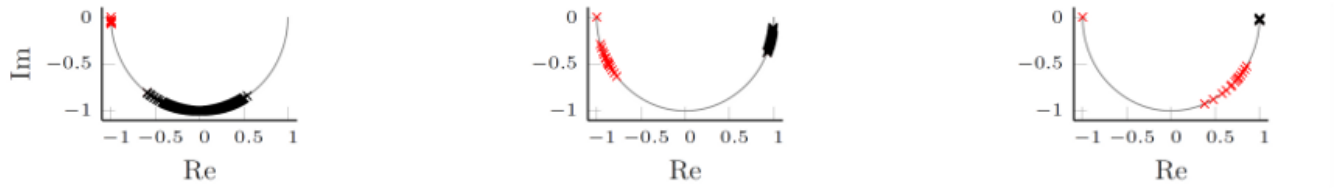


Reminds me of fractal visualizations (Courtesy of the globberingmattress blog)

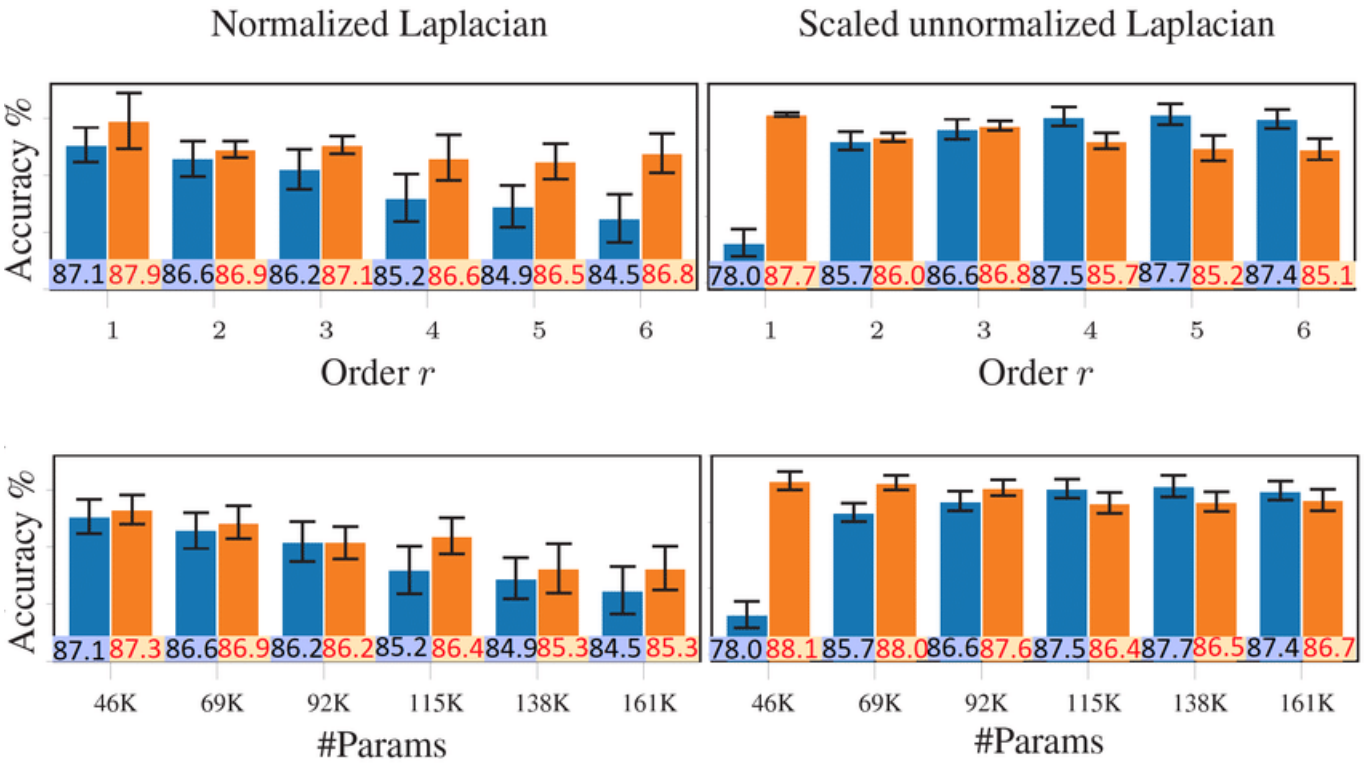
At a high level, the transform helps “zoom” into skewed datapoints as shown in this wacky GIF.

Cayley polynomials share similar properties with the Chebshev polynomials in ChebNets, including the useful notion of **localization** (note that in both approaches, “polynomials” refers to the function that acts as the kernel/filter).

Cayley has proven to perform better on a wide range of Graph Learning tasks due to their ability to **detect narrow frequency bands of importance during training, and to**



Thanks to a parameter (h) in the Cayley kernel/filter, smaller eigenvalues on the spectrum can be spread apart, allowing the **kernels to specialize in different frequencies.** This is visualized in the 3 diagrams from the original paper demonstrating the normalizing power of the parameter (h), where the marks indicate the eigenvalues (red being that which holds important info).

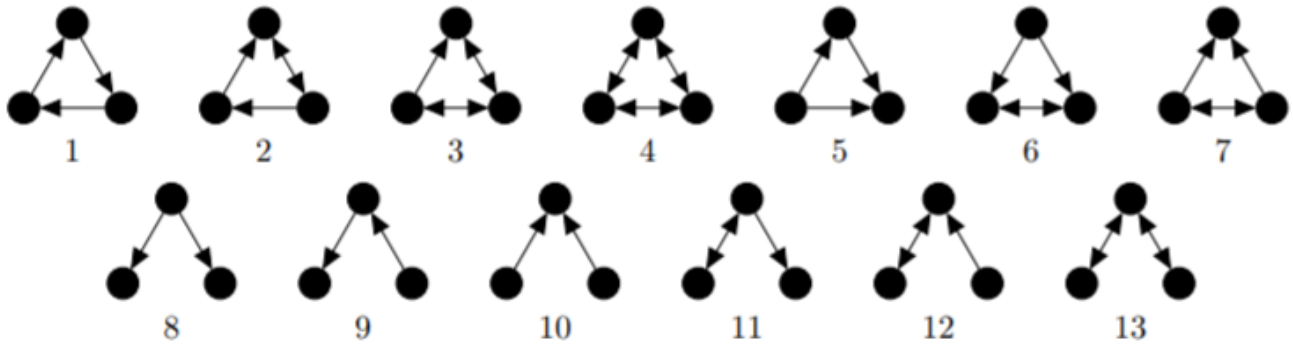


Cheb vs Cayley (Courtesy of ResearchGate)

Side by side, CayleyNets (orange) mostly outperformed ChebNets (blue), whilst requiring less parameters.

MotifNets

MotifNets was a different approach that built upon the original GCN model by introducing the notion of “motifs”. Essentially, the mode partitions input graphs into



The motifs used in the original paper (Courtesy of the MotifNet team)

These motifs are of size $x = 3$. Notice that the motifs take into **directionality of edges** into consideration, a detail in graph theory that has been omitted in previous graph learning approaches. This is especially useful for applications that have an intrinsic notion of direction such as social networks, transportation networks, and energy grids.

ChebNet is a instance of MotifNet with a single Laplacian of an undirected graph, in which a matrix of Chebyshev polynomials is used. Each convolutional layer of a MotifNet has a **multivariate matrix polynomial** (a fancy kernel where each element is a polynomial with multiple variables), which is applied to and learns from the motif's Laplacian matrices.

Despite the efficiency of ChebNet and GCN, both methods struggle when dealing with graphs containing **clustered eigenvalues**, a phenomenon typically concurring in community graphs. In this regard, MotifNet strives to fix the same weakness as CayleyNet.

Spectral methods overall

Robust, reliable, and researched, spectral convolutions kick-started interest in Graph Learning and Geometric Deep Learning as a whole. Even Yann Lecun and other researchers at the forefront of this niche have made contributions.

However, spectral methods have no shortage of weaknesses and drawbacks, but that's a topic for another post.

Spatial Graph Convolutional Networks

GraphSage — Hamilton et al

The beauty of GraphSage is its simplicity. The model is holds top-tier and remains competitive in terms of performance, even with newer or more powerful models. It is especially powerful because it scales well with large, **dense, homogenous, dynamic networks.** Created in part by the contributions of Jure Leskovec, who also contributed to various other algorithms in this article (including node2vec), GraphSage is one of many Graph Learning algorithms that have come out of SNAP, Stanford's Network Analysis Project.

At a high level, GraphSage has 3 steps:

Neighborhood sampling:

Start by finding the immediate neighborhood of the target node in a graph. **The depth k, is defined by the user, and determines how many "neighbors of neighbors" will be sampled.** This is operation is performed recursively, for a set number of steps.

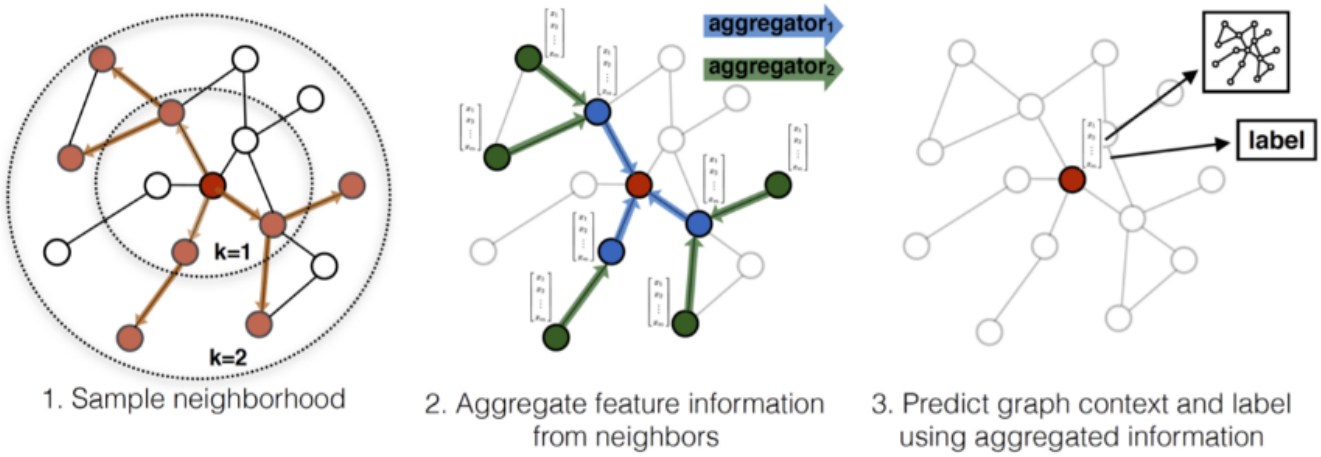
Aggregation:

After each node in the graph has sampled its respective neighborhood, we must **bring together all the features of the neighborhood nodes to the target node.** The original paper proposed 3 aggregation functions

- *Mean aggregation* — Averaging all the neighborhood node features (can be weighted average)
- *LSTM aggregation* — Using an LSTM cell to selectively aggregate neighborhood node features (ordered randomly)
- *Pooling aggregation* — Max pooling only takes the "highest" feature into consideration (performed the best in experiments)

Prediction:

determination. This is where the learning happens. This 3 step process is repeated for every node in the graph in a **supervised manner**. A visual representation to the approach is shown below:



GraphSage (Courtesy of the GraphSage research team)

Experiments showed that pooling aggregation performed best, as well as being the most efficient to compute. The effectiveness of GraphSage is why it is one of the few Graph Learning models currently implemented in a real-world application. Pintrest, the photo sharing website, currently uses GraphSage (albeit modified and renamed PinSage) to predict relevant photos based on the user's interests and search queries.

Mixture Model Networks (MoNet) — Monti et al

MoNet has seen much success in further research and development, with the original paper becoming the inspiration and spawn point for numerous approaches and architectures including Geodesic CNNs (GCNNs), Anistropic CNNs (ACNNs), Spline CNNs, and Diffusion CNNs.

The original MoNet paper had a 3-fold contribution

1. A generalization of various Graph Learning approaches, unifying spatial and spectral approaches

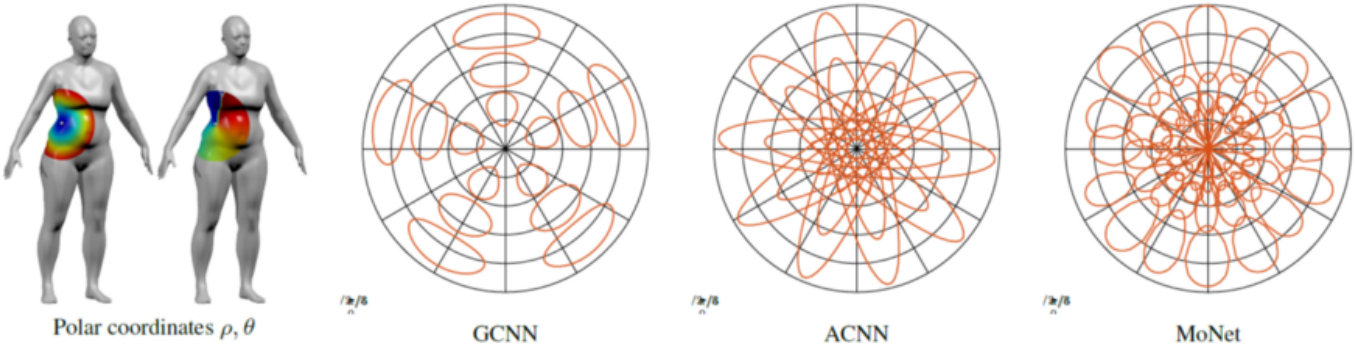
2. A new approach using parametric kernels, pseudo-coordinates, integrated with existing models (Anistropic CNN, Geodesic CNN,)

MoNet's generalization first considers variable ***x*** as a point in the manifold or node in a graph depending on the application, task, and input. Variables ***y*** is considered the neighboring nodes or points, which is associated with a vector of **pseudo-coordinates in d-dimensional space,** such that ***u(x, y)*** are the pseudo-coordinates of which there is a unique set for each neighbor of ***x***.

Each pseudo-coordinate is put through a **weighting function,** which replicates the effect of a traditional image convolution kernel whereby each value of the kernel is multiplied by the value that is currently in consideration. In the case of MoNet, the weighting function is a kernel with learnable parameters that operates on the pseudo-coordinates:

$$w_j(\mathbf{u}) = \exp\left(-\frac{1}{2}(\mathbf{u}-\boldsymbol{\mu}_j)^\top \boldsymbol{\Sigma}_j^{-1}(\mathbf{u}-\boldsymbol{\mu}_j)\right)$$

where ***Σj*** is a learnable ***d by d*** covariance matrix, and ***μj*** is a ***d by 1*** mean vector of the kernel. ***d*** is the dimensions of the pseudo-coordinate vectors (***u(x, y)***). These vector/pseudo-coordinate convolution kernels map out to some very interesting visualizations:



The normalized kernels applied to a manifold (Courtesy of the MoNet research team)

On the 3D models are local polar coordinates centered at a point on their manifolds. The circle diagrams (which represent the multi-chromatic parts of the models) are **patch**

This **kernel is learned only in MoNet, versus hard-coded in GCNNs and ACNNs.** The first thing one might notice is how much "smoother" the weights (red curves) are in MoNet, versus the more distinct (GCNN) and more integrated (ACNN) weights of the other approaches.

These kernel visualizations would change with each unique configuration of GCNNs and ACNNs. But MoNet, like GraphSage, can learn parameters. This flexibility is part of the reason why both allow for the **sharing of parameters cross-dataset.** This opens up the capacity for ideas like **Transfer Learning** to be applied to the geometric domain. The future of Graph Learning is looking pretty awesome!

Spatial methods overall

Spatial methods are alot easier to grasp, especially for the majority of people who don't have a strong intuition for linear algebra related operations like eigendecomposition, Fourier transforms, and other tough math concepts.

Lately, spatial methods have been garnering more and more momentum, the potential reasons for which are the topic for another post.

In Essence

During the 2010s to 2012s, due to the joint effort of the best researchers in Deep Learning including the likes of Yann Lecun (Image convolutions) and Geoff Hinton (Back-propagation), convolutional neural networks ignited a Deep Learning comeback that only they saw coming. **Effective and efficient graph convolutions have the potential to have that effect**, pushing the field of Geometric Deep Learning into the spotlight.

The differences between Spectral and Spatial convolutions have hopefully been elucidating, and ultimately, the two approaches can be characterized as having differences like so:

> ***markedly similar*** *to vanilla convolutions.*

Where one method follows textbook definitions and math to generalize a convolution, the other takes a stance from the graph theory perspective.

I've definitely missed a bunch of algorithms and models, especially since the recent explosion of interest in Geometric Deep Learning and Graph Learning has led to new contributions popping up in publications almost daily. Some interesting approaches include projects like Graph U-Net and Graph Markov Neural Networks. There are also existing models that I haven't gotten around to fully understanding yet; but when I do, I'll update this article.

Key Takeaways

- The purpose of graph convolutions is to **generalize the image convolution operation** to graphs so that we can achieve similar levels of performance and accuracy.
- Graph convolutions are different from image convolutions because **graphs as a data structure with non-euclidean properties are very different** from the set structure of an euclidean image.
- Graph Learning approaches are separated into 2 factions; s**pectral methods and spatial methods.** The classifications is determined by examining the use of eigendecomposition and related operations.
- Spectral methods attempt to use **concepts like signals to represent node features, and operations like the Fourier transform to aggregate node information** and make a prediction.
- Spatial methods use **operations like message passing and representation methods like pseudo-coordinates to aggregate information between nodes** and make a prediction.

> ***Up next, is a dive into recurrent graph and attention based methods*** 🔥

Need to see more content like this?

*All my content is on **my website** and all my projects are on **GitHub***

*I'm always looking to meet new people, collaborate, or learn something new so feel free to reach out to **flawnsontong1@gmail.com***

Upwards and onwards, always and only