Graph Neural Networks
Architectures, Fundamental Properties and Applications

Navid NaderiAlizadeh, Alejandro Ribeiro, Luana Ruiz, Zhiyang Wang

Web: gnn.seas.upenn.edu/aaai-2025/

Feb 26 2025

Graph Neural Networks
Architectures, Stability, and Transferability

Alejandro Ribeiro

Dept. of Electrical and Systems Engineering

University of Pennsylvania

Email: aribeiro@seas.upenn.edu Web: alelab.seas.upenn.edu

TH15: Graph Neural Networks: Architectures, Fundamental Properties and Applications
gnn.seas.upenn.edu/aaai-2025/

Feb 26 2025

Graph Neural Networks: Why?

How to use this site (I am not at Penn)

If you are a not a student at Penn, the instructors are honored that you consider the materials worth checking. We wish we had the time to work with you, alas, we do not. However, we think there is quite a lot that you can learn by watching the recorded video lectures and by working on the lab assignments. We have designed the materials with the goal of making them useable in self directed learning. How much we have succeeded at that is for you to decide.

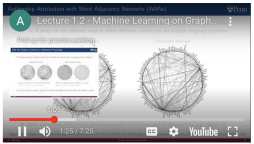
When you check the video lectures you will see that they come with a handout and a script. The handout and the script are designed to be used in conjunction with the videos. The videos are the union of the handout and the script. In addition to video lectures, you will find lab assignments and their solutions accessible through the lab webpage. The labs are designed with a complexity progression in mind. Start from Lab 1 if you have never worked in machine learning. Start from Lab 2 if you have ever encountered GNNs. Labs 3 and onwards are serious applications of GNNs to practical problems.

If you are hardcore and would rather read papers, this post has links to the papers that have inspired this course. These papers are part of the work on graph neural networks going on at Alelab. They are not a comprehensive literature review. You can find a little bit of that in this tutorial article in the signal processing magazine and in this more comprehensive review in the Proceedings of the IEEE.

If there is something you think we could to to help. We are happy to hear suggestions.

Video 1.2 – Machine Learning on Graphs: The Why

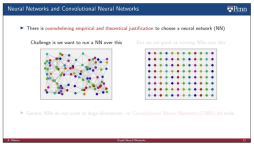
We care about GNNs because they enable machine learning on graphs. But why should we care about machine learning on graphs? We dwell here on the whys of machine learning on graphs. Why is it interesting? Why do we care? The reason we care is simple: Because graphs are pervasive in information processing.



• Covers Slides 6-10 in the handout.

Video 1.3 – Machine Learning on Graphs: The How

Having discussed the why, we tackle the how. How do we do machine learning on graphs? The answer to this question is pretty easy: We should use a neural network. We should do this, because we have overwhelming empirical and theoretical evidence for the value of neural networks. Understanding this evidence is one of the objectives of this course. But before we are ready to do that, there is a dealbreaker challenge potentially lurking in the shadows: Neural Networks must exploit structure to be scalable.

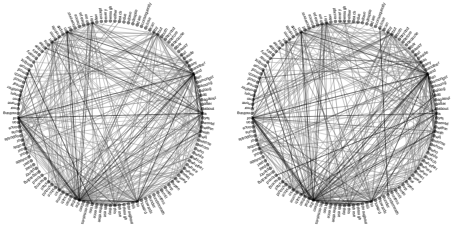


• Covers Slides 11-13 in the handout.

Machine Learning on Graphs: Why?

- Graphs are generic models of signal structure that can help to learn in several practical problems

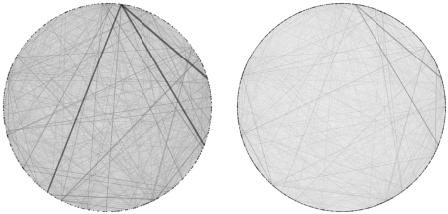
Authorship Attribution



Identify the author of a text of unknown provenance

Segarra et al '16, arxiv.org/abs/1805.00165

Recommendation Systems



Predict the rating a customer would give to a product

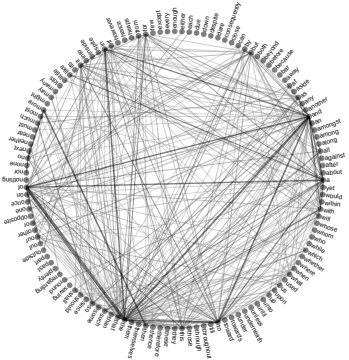
Ruiz et al '18, arxiv.org/abs/1903.12575

- In both cases there exists a graph that contains meaningful information about the problem to solve

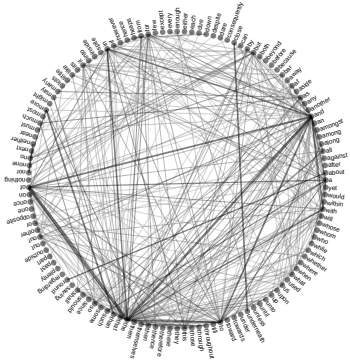
- Nodes represent different function words and edges how often words appear close to each other

  $\Rightarrow$ A proxy for the different ways in which different authors use the English language grammar

William Shakespeare



Christopher Marlowe



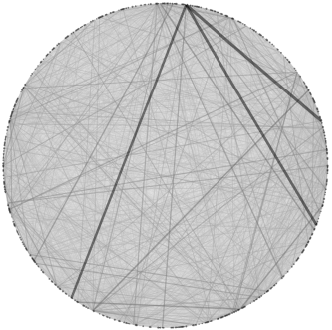
- WAN differences differentiate the writing styles of Marlowe and Shakespeare in, e.g., Henry VI

Segarra-Eisen-Egan-Ribeiro, *Attributing the Authorship of the Henry VI Plays by Word Adjacency,* Shakespeare Quarterly 2016, doi.org/10.1353/shq.2016.0024

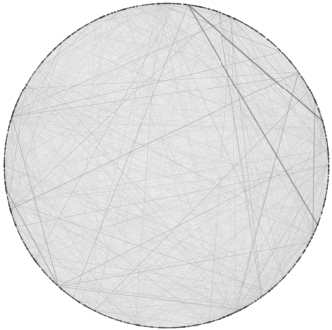
- Nodes represent different customers and edges their average similarity in product ratings

  $\Rightarrow$ The graph informs the completion of ratings when some are unknown and are to be predicted

Variation Diagram for Original (sampled) ratings



Variation Diagram for Reconstructed (predicted) ratings

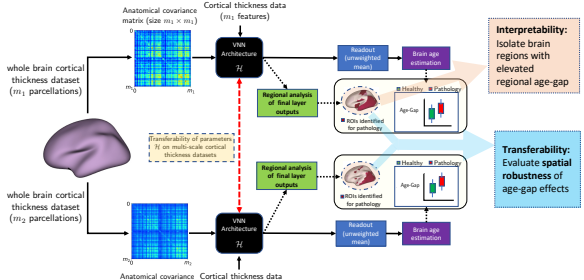


- Variation energy of reconstructed signal is (much) smaller than variation energy of sampled signal

Ruiz-Gama-Marques-Ribeiro, *Invariance-Preserving Localized Activation Functions for Graph Neural Networks*, arxiv.org/abs/1903.12575

Ageing is a risk factor for neurodegeneration and biological age (brain age) is elevated compared to chronological age in pathology.

Hence, **Age-Gap** (brain age – chronological age) is a biomarker of interest.



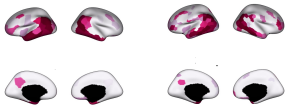
Cortical Thickness Brain Signals. GNN on anatomical covariance matrix leverages cortical thickness (CT) features to predict brain age.

Regional age-gap is defined by the difference between GNN prediction and outputs at the final layer of GNN.

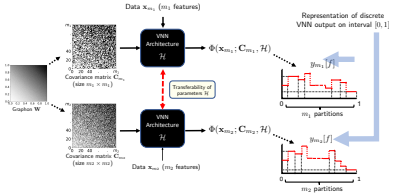
Elevated brain age gap effect is driven by regional age-gap effects in impacted regions.

Interpretable regional profile to elevated brain age.

Regions with elevated age-gap in Alzheimer's Disease



GNN can be transferred across different graphs



Sihag-Mateos-McMillan-Ribeiro, *coVariance Neural Networks.*, arxiv.org/abs/2205.15856

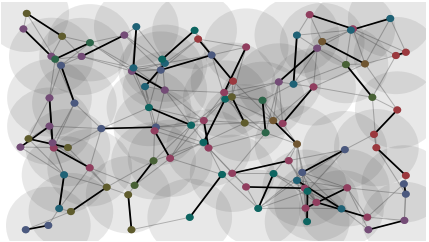
- Graphs are more than data structures $\Rightarrow$ They are models of physical systems with multiple agents

Decentralized Control of Autonomous Systems

Coordinate a team of agents without central coordination

Tolstaya et al '19, arxiv.org/abs/1903.10527

Wireless Communications Networks



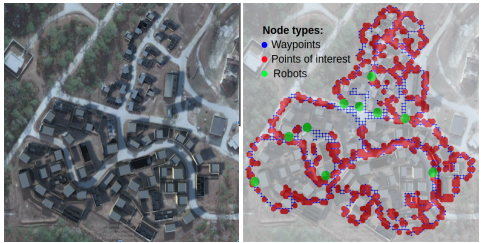
Manage resources in wireless communications

Eisen-Ribeiro '19, arxiv.org/abs/1909.01865

- The graph is the source of the problem $\Rightarrow$ Challenge is that goals are global but information is local

- Graphs are more than data structures $\Rightarrow$ They are models of physical systems with multiple agents

Decentralized Control of Autonomous Systems



Collaborative navigation of roads with a team of agents

Tolstaya et al '21, arxiv.org/abs/2011.01119

Wireless Communications Networks

Mobile infrastructure on demand to support a task team

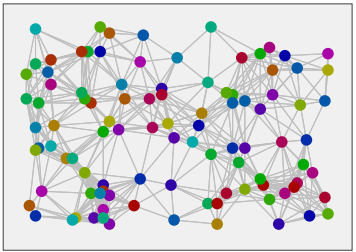
Mox et al '22, arxiv.org/abs/2112.07663

- The graph is the source of the problem $\Rightarrow$ Challenge is that goals are global but information is local

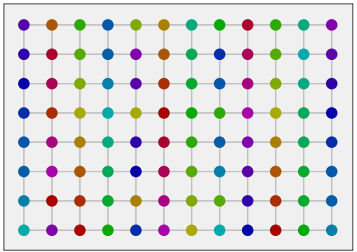
Machine Learning on Graphs: How?

- There is overwhelming empirical and theoretical justification to choose a neural network (NN)

Challenge is we want to run a NN over this



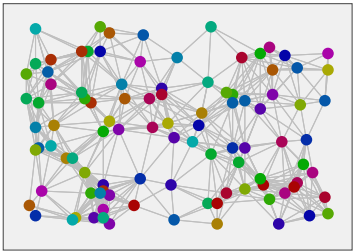
But we are good at running NNs over this



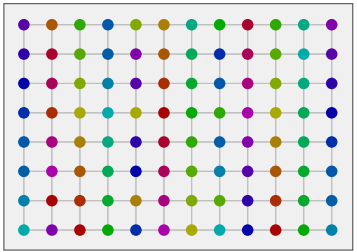
- Generic NNs do not scale to large dimensions $\Rightarrow$ Convolutional Neural Networks (CNNs) do scale

- CNNs are made up of layers composing convolutional filter banks with pointwise nonlinearities

Process graphs with graph convolutional NNs



Process images with convolutional NNs



- Generalize convolutions to graphs $\Rightarrow$ Compose graph filter banks with pointwise nonlinearities
- Stack in layers to create a graph (convolutional) Neural Network (GNN)

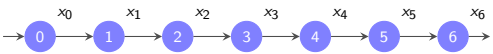
Convolutions in Time, in Space, and on Graphs

- How do we generalize convolutions in time and space to operate on graphs?

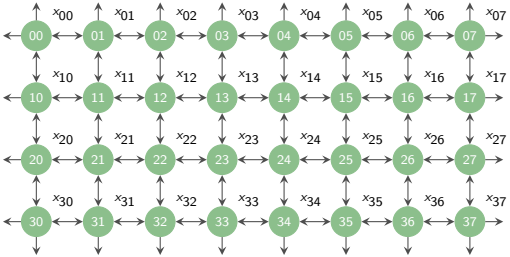
  $\Rightarrow$ Even though we do not often think of them as such, convolutions are operations on graphs

- We can describe discrete time and space using graphs that support time or space signals

Description of time with a directed line graph



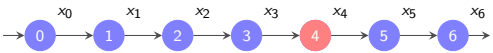
Description of images (space) with a grid graph



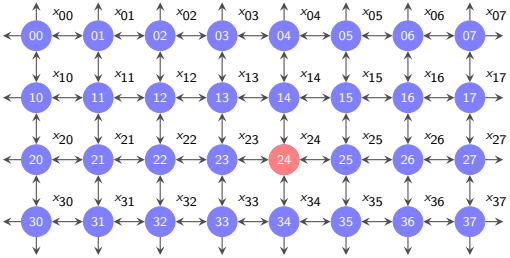
- Line graph represents adjacency of points in time. Grid graph represents adjacency of points in space

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph



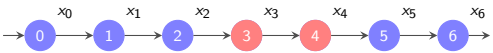
Description of images (space) with a grid graph



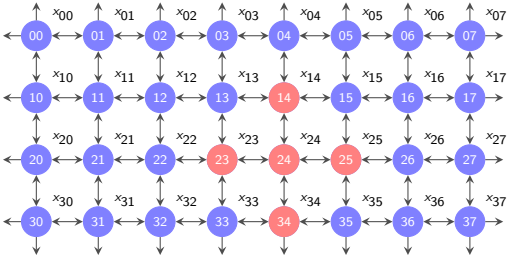
- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0\mathbf{x} + h_1 \mathbf{S}^1\mathbf{x} + h_2 \mathbf{S}^2\mathbf{x} + h_3 \mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k\mathbf{x}$

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph



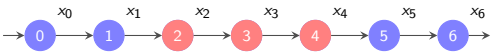
Description of images (space) with a grid graph



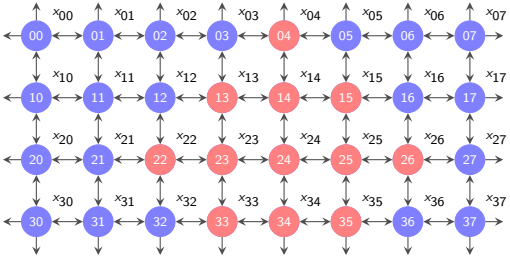
- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0\mathbf{x} + h_1 \mathbf{S}^1\mathbf{x} + h_2 \mathbf{S}^2\mathbf{x} + h_3 \mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k\mathbf{x}$

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph



Description of images (space) with a grid graph

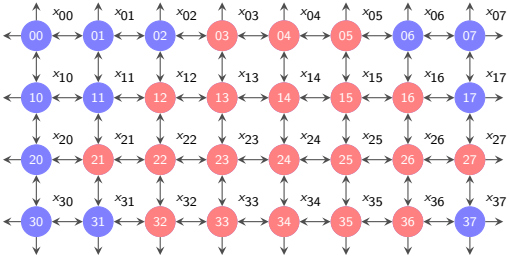


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0\mathbf{x} + h_1 \mathbf{S}^1\mathbf{x} + h_2 \mathbf{S}^2\mathbf{x} + h_3 \mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k\mathbf{x}$

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph

Description of images (space) with a grid graph

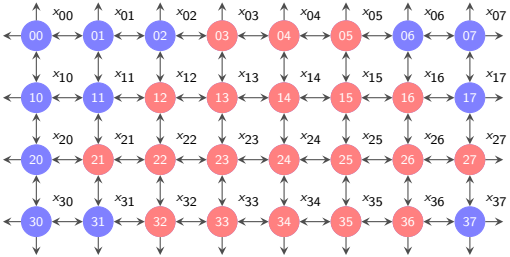


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0\mathbf{x} + h_1 \mathbf{S}^1\mathbf{x} + h_2 \mathbf{S}^2\mathbf{x} + h_3 \mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k\mathbf{x}$

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph

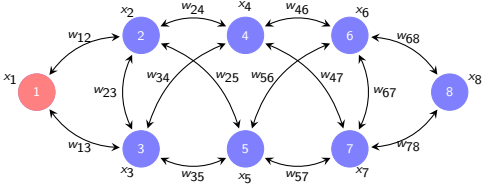
Description of images (space) with a grid graph



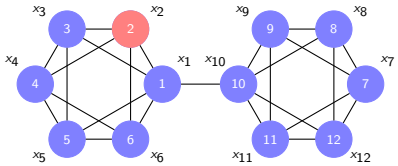
- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0\mathbf{x} + h_1 \mathbf{S}^1\mathbf{x} + h_2 \mathbf{S}^2\mathbf{x} + h_3 \mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k\mathbf{x}$

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph

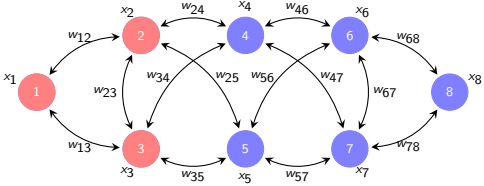


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

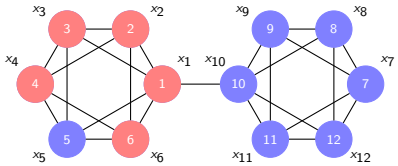
- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph

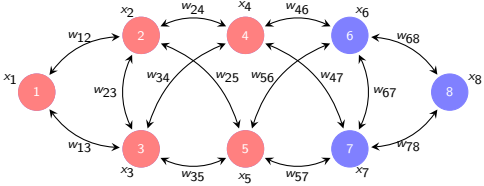


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

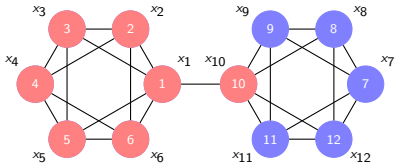
- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph

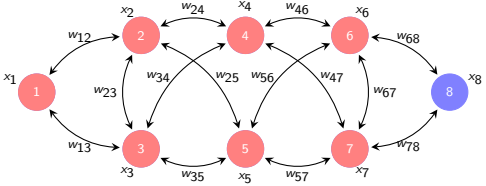


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

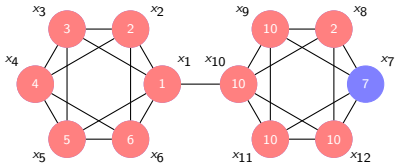
- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph

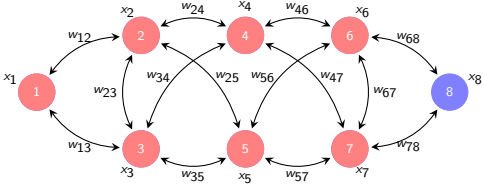


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0\mathbf{x} + h_1 \mathbf{S}^1\mathbf{x} + h_2 \mathbf{S}^2\mathbf{x} + h_3 \mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k\mathbf{x}$

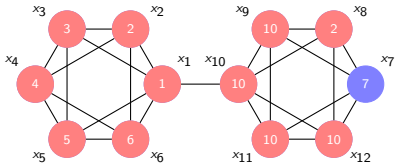
- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



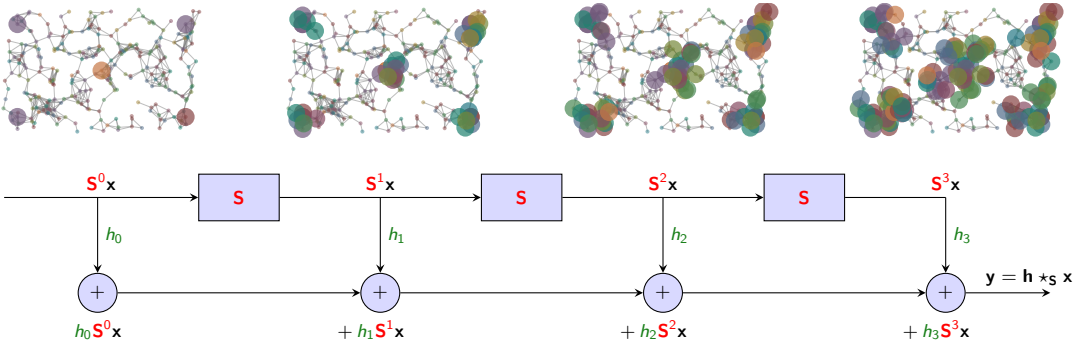
Another signal supported on another graph



- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0\mathbf{x} + h_1 \mathbf{S}^1\mathbf{x} + h_2 \mathbf{S}^2\mathbf{x} + h_3 \mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k\mathbf{x}$

- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

- A graph convolution is a weighted linear combination of the elements of the diffusion sequence
- Can represent graph convolutions with a shift register $\Rightarrow$ Convolution $\equiv$ Shift. Scale. Sum



Definition (Convolution)

A convolutional filter is a polynomial on a shift operator with coefficients h_k $\Rightarrow \mathbf{z} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

- It is the same algebraic object whether we consider time, space, or graphs
- They all have compositionality (operator powers) and some kind of equivariance
- They all admit a frequency representation

  $\Rightarrow$ Filters are pointwise operators in the eigenvector basis of the shift operator

Parada Mayorga-Ribeiro , *Algebraic Neural Networks: Stability to Deformations*, arxiv.org/abs/2009.01433

Definition (Algebraic Convolutions with Multiple Features)

Input signal $\mathbf{X} \in \mathbb{R}^{N \times F}$ with F features. Output signal $\mathbf{Z} \in \mathbb{R}^{N \times G}$ with G features. Filter coefficients $\mathbf{H}_k$ are $F \times G$ matrices. The convolutional filter with coefficients $\mathbf{H}_k$ is

$$\mathbf{Z} = \sum_{k=0}^{\infty} \mathbf{S}^k \times \mathbf{X} \times \mathbf{H}_k$$

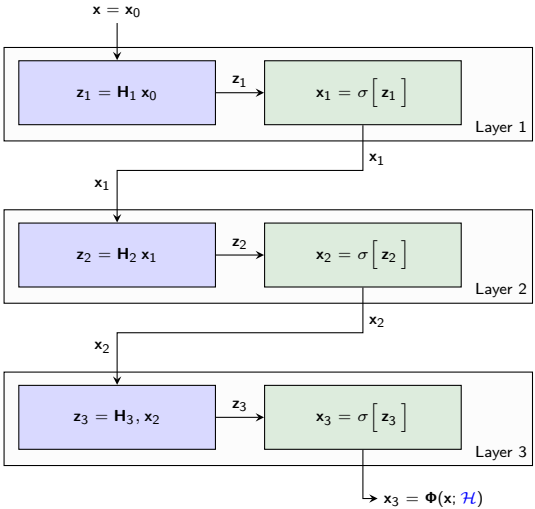
- It has the same algebraic structure of a regular filter with scalar coefficients.
- Retains compositionality, equivariance, and existence of a frequency representation
- Filters with multiple features are more expressive. The ones we use to build GNNs and CNNs

Convolutional Neural Networks and Graph Neural Networks

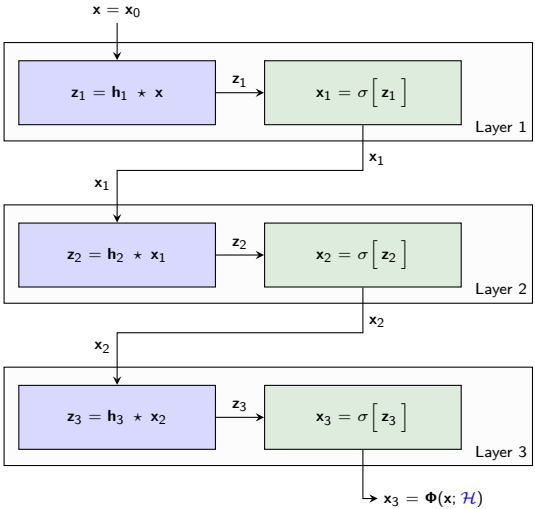
- CNNs and GNNe are minor variations of linear convolutional filters

  $\Rightarrow$ Compose filters with pointwise nonlinearities and compose these compositions into several layers

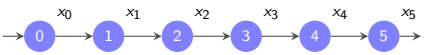
- A neural network composes a cascade of layers
- Each of which are themselves compositions of linear maps with pointwise nonlinearities
- Does not scale to large dimensional signals $\mathbf{x}$



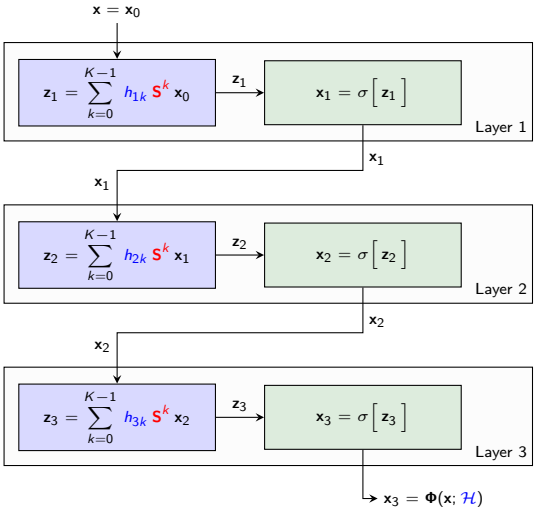
- A convolutional NN composes a cascade of layers
- Each of which are themselves compositions of convolutions with pointwise nonlinearities
- Scales well. The Deep Learning workhorse
- A CNNs are minor variation of convolutional filters
  - $\Rightarrow$ Just add nonlinearity and compose
  - $\Rightarrow$ They scale because convolutions scale



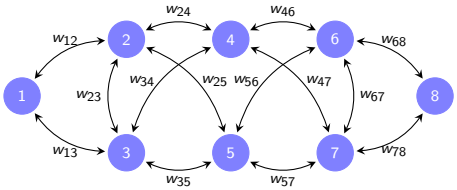
- Those convolutions are polynomials on the adjacency matrix of a line graph

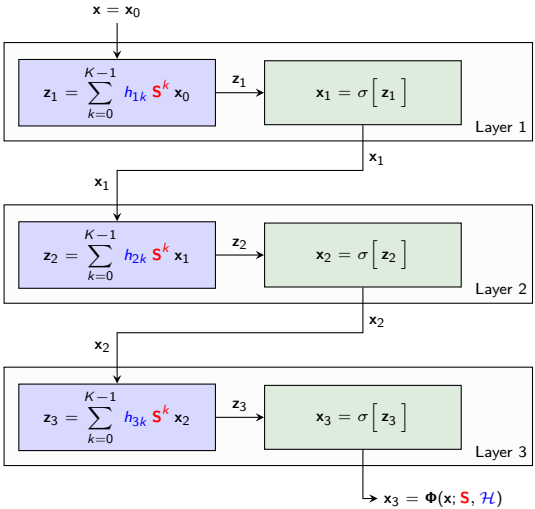


- Just another way of writing convolutions and Just another way of writing CNNs
- But one that lends itself to generalization



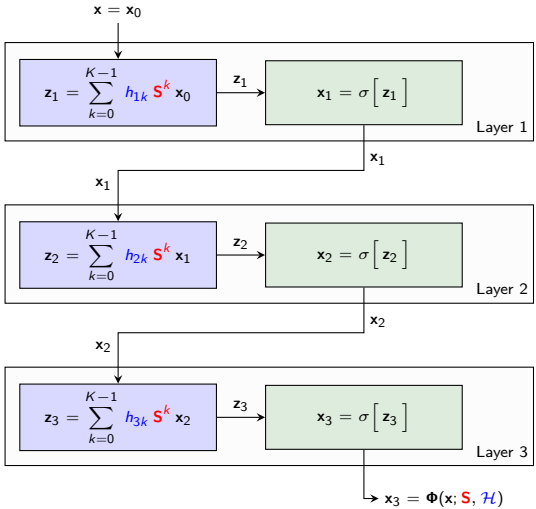
- The graph can be any arbitrary graph
- The polynomial on the matrix representation $\mathbf{S}$ becomes a graph convolutional filter





Gama-Marques-Leus-Ribeiro, *Convolutional Neural Network Architectures for Signals Supported on Graphs,* TSP 2019, arxiv.org/abs/1805.00165

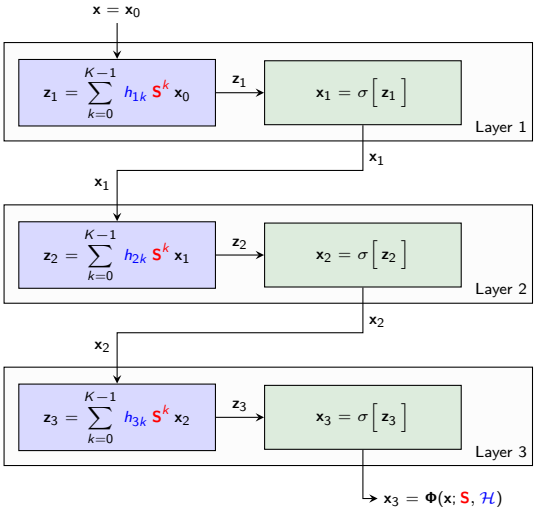
- A graph NN composes a cascade of layers
- Each of which are themselves compositions of graph convolutions with pointwise nonlinearities
- A NN with linear maps restricted to convolutions
- Recovers a CNN if $\mathbf{S}$ describes a line graph



Gama-Marques-Leus-Ribeiro, *Convolutional Neural Network Architectures for Signals Supported on Graphs,* TSP 2019, arxiv.org/abs/1805.00165

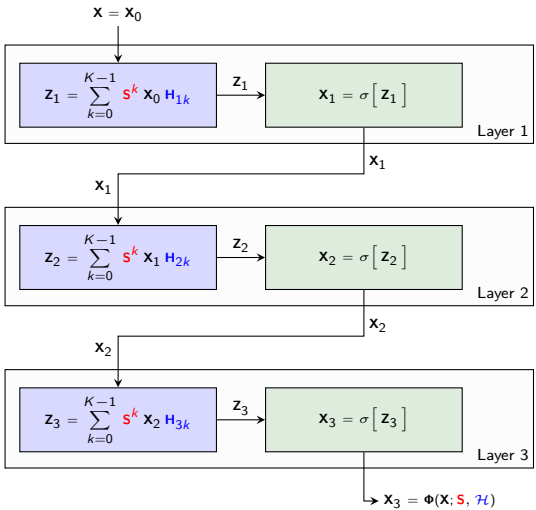
- There is growing evidence of scalability.
- A GNN is a minor variation of a graph filter

  $\Rightarrow$ Just add nonlinearity and compose
- Both are scalable because they leverage the signal structure codified by the graph



Gama-Marques-Leus-Ribeiro, *Convolutional Neural Network Architectures for Signals Supported on Graphs,* TSP 2019, arxiv.org/abs/1805.00165

- In practice we use layers with multiple features
- This is to increase representation power but it does not affect our fundamental observations



Gama-Marques-Leus-Ribeiro, *Convolutional Neural Network Architectures for Signals Supported on Graphs,* TSP 2019, arxiv.org/abs/1805.00165

Collaborative Filtering with Graph Neural Networks

Juan Elenter, Tatiana Guevara, Ignacio Hounie, Charilaos Kanatsoulis, and Alejandro Ribeiro*

March 5, 2023

1 Collaborative Filtering

The objective of this lab is to design a recommendation system that predicts the ratings that customers would give to a certain product. Say, the rating that a moviegoer would give to a specific movie, or the rating that an online shopper would give to a particular offering. A possible approach to making these predictions is to leverage the ratings that customers have given to this or similar products in the past. This is called collaborative filtering.

A schematic representation of collaborative filtering is shown in Figure 1. The underlying assumption is that there is a true set of ratings that different customers would give to specific products. These ratings remain unobserved and are denoted by $\mathbf{X}$ in Figure 1. What we *do* have available are a subset of these ratings. They are represented by $\mathbf{X}$ in Figure 1 where all of the missing ratings are represented by a blank space. This is a reasonable model of reality. Each of us has seen a small number of movies or bought a small number of offerings. Thus, the ratings matrix $\mathbf{X}_o$ contains only a few entries that correspond to rated products. Our goal is to recover estimates $\mathbf{Y}$ of the unobserved ratings $\mathbf{X}$.

*In alphabetical order.

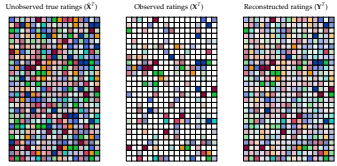


Figure 1. Recommendation with Collaborative Filtering.

As a specific example, we use the MovieLens-100k dataset. The MovieLens-100k dataset consists of ratings given by U users to P movies (products). The existing movie ratings are integer values between 1 and 5. Therefore, the data are represented by a $U \times P$ matrix X where x_{up} is the rating that user u gives to movie p. If user u has not rated movie p, we adopt the convention that $x_{up} = 0$. We see that each row of this matrix corresponds to a vector of ratings $\mathbf{x}_u$ of a particular user.

1.1 Product Similarity Graph

To build the collaborative filtering system, we use the rating history of all movies and all users to compute a graph of product similarities. This is a graph in which nodes p represent different movies and weighted edges S_{pq} denote similarities between products p and q. The edges of the graph are grouped in the adjacency matrix $\mathbf{S}$.

To compute the entries S_{pq} of the product similarity graph we use the raw $U \times P$ movie rating matrix to evaluate crosscorrelations between movie ratings of products p and q. To make matters simpler we have constructed this graph already and are making it available as part of the dataset.

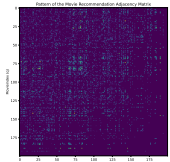


Figure 2. Adjacency Matrix of the Movie Similarity Graph. Brighter dots correspond to pairs of movies that different watchers tend to score with similar ratings.

Task 1 Download the movie rating data to your computer and upload the data "movie_data_numpy.p" to this processing environment. Plot the adjacency matrix **S** as an image. ■

Success in Task 2 must have produced the plot in Figure 2. In this figure each bright dot corresponds to a large entry $S(p,q)$. This denotes a pair of movies to which watchers tend to give similar scores. For instance, say that when someone scores "Star Wars IV" highly, they are likely to score "Star Wars V" highly and that the converse is also true; poor scores in one correlate with poor scores in the other. The entry $S($"Star Wars IV", "Star Wars V"$)$ is large because the crosscorrelation between the scores of these entries is high.

Fainter entries $S(p,q)$ denote pairs of movies with less socre correlation. Perhaps between "Star Wars" and "Star Trek" which have overlapping but not identical fan bases. Dark entries $S(p,q)$ correspond to pairs movies with no correlation between audience scores. Say when p is the index of "Star Wars" and q is the index of "Little Miss Sunshine."

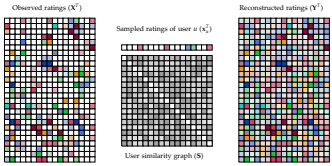


Figure 3. Reconstruction of Movie Ratings with a Movie Similarity Matrix

1.2 Rating Signals

The vector of ratings $\mathbf{x}_u$ of a particular user is interpreted as a signal supported on the graph. That is, a signal in which the pth component x_{up} is associated with node p. In this context, the weights of the product similarity graph become an expectation of similarity between ratings x_{up} and x_{uq}. If S_{pq} is large we expect these ratings to be similar. If S_{pq} is small we have no expectation of proximity or not between them.

We then have a system with the architecture shown in Figure 3. Rating signals $\mathbf{x}_u$ of individual users are extracted from the raw rating matrix and are interpreted as signals supported on the graph **S** that we loaded in Task 2. We want to leverage the graph **S** to make rating predictions $\mathbf{y}_u$ for this particular user.

We will, more precisely, develop and evaluate a graph neural network (GNN) for making these rating predictions.

1.3 Rating Data Format and Rating Loss

To train the collaborative filtering system we use rating histories to create a dataset with entries $(\mathbf{x}_n, y_n, p_n)$. In these entries $\mathbf{x}_n$ is a vector that contains the ratings of a particular user, y_n is a scalar that contains a rating that we want to predict, and p_n is the index of the movie (product) that corresponds to the rating y_n. To evaluate this collaborative filtering system we use rating histories to create a dataset with entries having the same format. Both of these datasets can be constructed from the raw $U I \times P$ movie rating matrix, but to make matters simpler we have constructed them already and are making them available as part of the dataset.

If we have a function $\hat{\mathbf{y}}_n = \mathbf{\Phi}(\mathbf{x}_n; \mathcal{H})$ that makes rating predictions out of availbale ratings, we can evaluate the goodness of this function with the squared loss

$$\ell\big(\mathbf{\Phi}(\mathbf{x}_n; \mathcal{H}), y_n\big) \;=\; \Big[\big(\hat{\mathbf{y}}_n\big)_{p_n} - y_n\Big]^2 \;=\; \Big[\big(\mathbf{\Phi}(\mathbf{x}_n; \mathcal{H})\big)_{p_n} - y_n\Big]^2. \quad (1)$$

Notice that in this expression the function $\hat{\mathbf{y}}_n = \mathbf{\Phi}(\mathbf{x}_n; \mathcal{H})$ makes predictions for all movies. However, we isolate entry p_n and compare it against the rating y_n. We do this, because the rating y_n of movie p_n is the one we have available in the training or test sets.

We remark the fact that the function $\hat{\mathbf{y}} = \mathbf{\Phi}(\mathbf{x}; \mathcal{H})$ makes predictions for all movies is important during operation. The idea of the recommendation system is to identify the subset of products that the customer would rate highly. They are the ones that we will recommend. This is why we want a system that has a graph signal as an output even though the available dataset has scalar outputs.

Task 2 Write a function to evaluate the training loss in (1). ■

2 Graph Convolutions

Let $\mathbf{S}$ denote a matrix representation of a graph. Supported on the nodes of the graph we are given a graph signal $\mathbf{x}$. We also consider a set of K

coefficients h_k from $k = 0$ to $k = K - 1$. A graph convolutional filter is a linear map acting on $\mathbf{x}$ defined as a polynomial on the matrix representation of the graph with coefficients h_k,

$$\mathbf{z} \;=\; \mathbf{h} *_{\mathbf{S}} \mathbf{x} \;=\; \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}. \quad (2)$$

Graph convolutions generalize convolutions in time to graphs. That this is true can be seen if we represent time with a directed line graph. Considering (2) for the particular case in which $\mathbf{S}$ is the adjacency matrix of this line graph, the product $\mathbf{S}^k\mathbf{x}$ results in a k-shift of the time signal $\mathbf{x}$. For this reason we sometimes refer to $\mathbf{S}$ as a shift operator. We also point out that although we work with an adjacency matrix in this lab any matrix representation of the graph can be used in (2).

One advantage that graph filters share with time convolutions is their locality. To see this, define the diffusion sequence as a collection of graph signals $\mathbf{u}_k = \mathbf{S}^k\mathbf{x}$ and rewrite the filter in (2) as,

$$\mathbf{z} \;=\; \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x} \;=\; \sum_{k=0}^{K} h_k \mathbf{u}_k \quad (3)$$

It is ready to see that the diffusion sequence is given by the recursion $\mathbf{z}_k = \mathbf{S}\mathbf{z}_{k-1}$ with $\mathbf{z}_0 = \mathbf{x}$. Further observing that $S_{ij} \neq 0$ only when the pair (i, j) is an edge of the graph, we see that the entries of the diffusion sequence satisfy

$$u_{k,i} \;=\; \sum_{j:(i,j)\in\mathcal{E}} S_{ij} u_{k-1,j}. \quad (4)$$

We can therefore interpret graph filters as operators that propagate information through adjacent nodes. This is analogous to the propagation of information in time with the application of time shifts. The locality of graph convolutions is one of the motivations for their use in the processing of information supported on graphs. The other reason is their *equivariance to permutations*.

Because it aggregates with a weighted sum the information from neighboring nodes, the operation in (4) is sometimes called an aggregation information. Because it aggregates at node i information that is passed from adjacent nodes j, we sometimes say that graph filters are message-passing architectures and the GNNs that are derived from them are called message passing GNNs.

2.1 Graph Convolutions with Multiple Features

To increase the representation power of graph filters we extend them to add multiple features. In these filters the input is a matrix $\mathbf{X}$ and the output is another matrix matrix $\mathbf{Y}$. The filter coefficients are matrices $\mathbf{H}_k$ and the filter itself is a generalization of (2) in which the matrices $\mathbf{H}_k$ replace the scalars h_k,

$$\mathbf{Z} = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X} \mathbf{H}_k. \tag{5}$$

In (5), the input feature matrix $\mathbf{X}$ has dimension $N \times F$ and the output feature matrix $\mathbf{Y}$ has dimension $N \times G$. This means that each of the F columns of $\mathbf{X}$ represents a separate input feature whereas each of the G columns of $\mathbf{Y}$ represents an output feature. To match dimensions, the filter coefficient matrices $\mathbf{H}_k$ must be of dimension $F \times G$.

Other than the fact that it represents an input-output relationship between matrices instead of vectors, (5) has the same structure of (2).

In particular, we can define a diffusion sequence $\mathbf{U}_k$ through the recursion $\mathbf{U}_k = \mathbf{S}\mathbf{U}_{k-1}$ and rewrite (5) as

$$\mathbf{Z} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{X} = \sum_{k=0}^{K} h_k \mathbf{U}_k. \tag{6}$$

This is worth remarking because we can write the diffusion sequence as a message passing aggregation operation. Indeed, if $\mathbf{u}_{k,i}$ is the ith row of $\mathbf{U}_k$ we can write the diffusion sequence recursion as

$$\mathbf{u}_{k,i} = \sum_{j:(i,j)\in\mathcal{E}} S_{ij} \mathbf{u}_{k-1,j}. \tag{7}$$

In (7) nodes j in the neighborhood of i pass the message $\mathbf{u}_{k-1,j}$. Node i aggregates these messages to create the updated message $\mathbf{u}_{k,i}$ that it passes on to its neighbors.

Task 3 Write a function that implements a graph filter. This function takes as inputs the shift operator $\mathbf{S}$, the filter coefficients $\mathbf{H}_k$ and the input signal $\mathbf{X}$. To further improve practical performance we add a bias term $\mathbf{B}$ to the filter operation. That is, we refine (5) with the opearation,

$$\mathbf{Z} = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X} \mathbf{H}_k + \mathbf{B}. \tag{8}$$

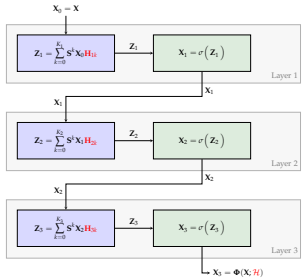


Figure 4. A Graph Neural Network (GNN) with three layers. A GNN is a composition of layers, each of which is itself the composition of a linear graph filter with a pointwise nonlinearity. [cf. (9)].

The bias $\mathbf{B}$ is also passed as a parameter to the filter function. ■

Task 4 The graph filter implemented in Task 3 is not a GNN but it is not a bad parameterization for making movie recommendations. Train a graph filter to predict movie ratings. Plot the evolution of the training loss and evaluate the loss in the test dataset. To obtain a good loss we need to experiment with the length of the filter – the number of filter taps K.

A graph filter is sometimes called a linear GNN. It is a GNN that does not use nonlinear opeartions.

3 Graph Neural Networks

Graph Neural Networks (GNNs) are information processing architectures made up of a composition of layers, each of which is itself the composition of a linear graph filter with a pointwise nonlinearity.

For a network with a given number of layers L we define the input output relationship through the recursion

$$\mathbf{X}_\ell = \sigma\Big(\mathbf{Z}_\ell\Big) = \sigma\left(\sum_{k=0}^{K_\ell} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\right), \tag{9}$$

In this recursion the output of Layer $\ell-1$ is $\mathbf{X}_{\ell-1}$ and it is recast as an input to Layer ℓ. In this layer, the input $\mathbf{X}_{\ell-1}$ is processed with a graph filter to produce the intermediate output $\mathbf{Z}_\ell$. The coefficients of this graph filter are the matrices $\mathbf{H}_{\ell k}$. This intermediate output is processed with a pointwise nonlinearity σ to produce the output $\mathbf{X}_\ell$ of Layer ℓ. That the nonlinear operation is pointwise means that it is acting separately on each entry of $\mathbf{Z}_\ell$.

To complete the recursion we redefine the input $\mathbf{X}$ as the output of Layer 0, $\mathbf{X}_0 = \mathbf{X}$. The output of the neural network is the output of layer L, $\mathbf{X}_L = \mathbf{Phi}(\mathbf{X}; \mathcal{H})$. In this notation $\mathcal{H}$ is the tensor $\mathcal{H} := [\mathbf{H}_{11}, \ldots, \mathbf{H}_{LK_L}]$ that groups all of the filters that are used at each of the L layers.

A graph neural network with three layers is depicted in Figure 4.

3.1 Graph Neural Network Specification

To specify a GNN we need to specify the number of layers L and the characteristics of the filters that are used at each layer. The latter are the number of filter taps K_ℓ and the number of features F_ℓ at the output of the layer. The number of features F_0 must match the number of features at the input and the number of features F_L must match the number of features at the output. Observe that the number of features at the output of Layer $(\ell-1)$ determines the number of features at the input of Layer ℓ. Then, the filter coefficients at Layer ℓ are of dimension $F_{\ell-1} \times F_\ell$.

Task 5 Program a class that implements a GNN with L layers. This class receives as initialization parameters a GNN specification consisting of the number of layers L and vectors $[K_1, \ldots, K_L]$ and $[F_0, F_1, \ldots, F_L]$ containing the number of taps and the number of features of each layer.

Endow the class with a method that takes an input feature $\mathbf{X}$ and produces the corresponding output feature $\mathbf{Phi}(\mathbf{X}; \mathcal{H})$. ■

Task 6 Train a GNN to predict movie ratings. Plot the evolution of the training loss and evaluate the loss in the test dataset. To obtain a good loss we need to experiment with the number of layers and the number of filter taps per layer.

Equivariance and Stability Properties of GNNs

Gama-Bruna-Ribeiro, *Stability Properties of Graph Neural Networks*, TSP 2020, arxiv.org/abs/1905.04497

Gama-Isufi-Leus-Ribeiro, *Graphs, Convolutions, and Neural Networks: From Graph Filters to Graph Neural Networks*, SPMag 2020, arxiv.org/abs/2003.03777

Ruiz-Gama-Ribeiro, *Graph Neural Networks: Architectures, Stability and Transferability,* PIEEE 2021 arxiv.org/abs/2008.01767

Fact 1

Graph filters and GNNs "work." Outperform general linear transforms and fully connected NNs.

Fact 2

GNNs outperform graph filters in most learning tasks.

Fact 1: Graph Filters and GNNs are Permutation Equivariant

Graph filters and GNNs leverage symmetries of graph signals

Fact 2

GNNs outperform graph filters in most learning tasks.

Fact 1: Graph Filters and GNNs are Permutation Equivariant

Graph filters and GNNs leverage symmetries of graph signals

Fact 2: Stability Properties of GNNs

GNNs can be simultaneously discriminative and stable to deformations. Graph filters cannot.

Fact 1: Graph Filters and GNNs are Permutation Equivariant

Graph filters and GNNs leverage symmetries of graph signals

- It is equally ready to show that GNNs are also equivariant to permutations of the input signals

Theorem (Permutation equivariance of graph neural networks)

Consider consistent permutations of the shift operator $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{S}\mathbf{P}$ and input signal $\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{x}$. Then

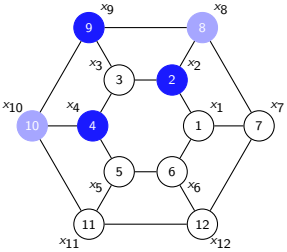
$$\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{H}) = \mathbf{P}^T \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$$

- Relabeling the input signal results in a consistent relabeling of the output signal

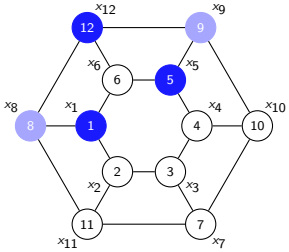
- Graph filters and GNNs, perform label-independent processing of graph signals

$\Rightarrow$ Permute input and shift $\equiv$ Relabel input $\Rightarrow$ Permute output $\equiv$ Relabel output

Graph signal $\mathbf{x}$ Supported on $\mathbf{S}$



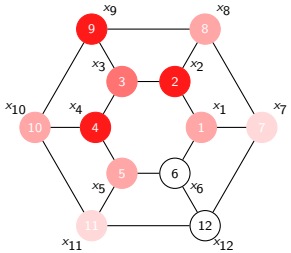
Graph signal $\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{x}$ supported on $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{S}\mathbf{P}$



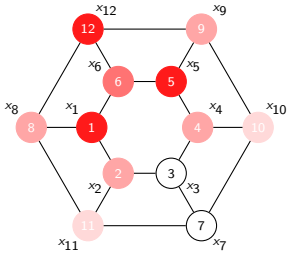
- Graph filters and GNNs, perform label-independent processing of graph signals

$\Rightarrow$ Permute input and shift $\equiv$ Relabel input $\Rightarrow$ Permute output $\equiv$ Relabel output

GNN output $\Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$ supported on $\mathbf{S}$

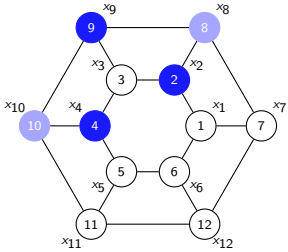


GNN $\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{H}) = \mathbf{P}^T \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$ on $\hat{\mathbf{S}} = \mathbf{P}^T \mathbf{S} \mathbf{P}$

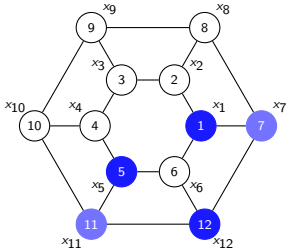


- Graph filters and GNNs exploit permutation symmetries of graphs and graph signals
- By symmetry we mean that the graph can be permuted onto itself $\Rightarrow \mathbf{S} = \mathbf{P}^T\mathbf{S}\mathbf{P}$
- Equivariance theorem implies $\Rightarrow \Phi\Big(\mathbf{P}^T\mathbf{x};\, \mathbf{S}, \mathcal{H}\Big) = \Phi\Big(\mathbf{P}^T\mathbf{x};\, \mathbf{P}^T\mathbf{S}\mathbf{P}, \mathcal{H}\Big) = \mathbf{P}^T\Phi\Big(\mathbf{x};\, \mathbf{S}, \mathcal{H}\Big)$

From observing $\mathbf{x}$ supported on $\mathbf{S}$



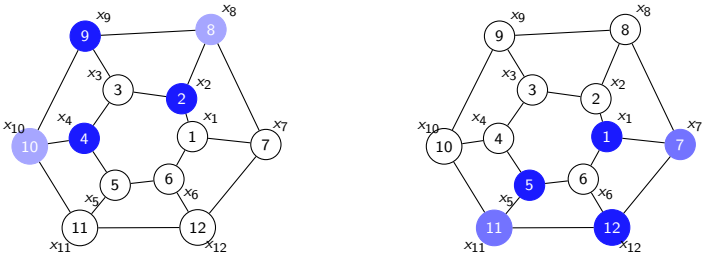
Learn to process $\mathbf{P}^T\mathbf{x}$ supported on $\mathbf{S} = \mathbf{P}^T\mathbf{S}\mathbf{P}$



Fact 2: Stability Properties of GNNs

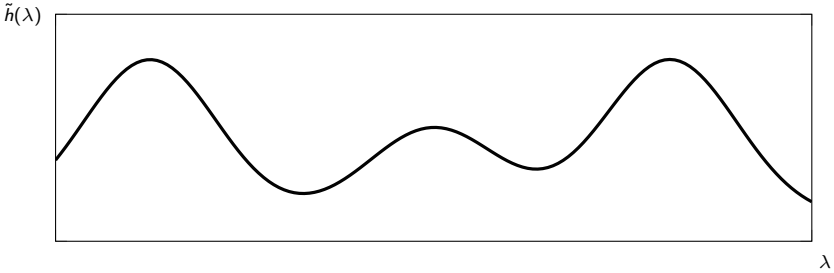
GNNs can be simultaneously discriminative and stable to deformations. Graph filters cannot.

- Graph not symmetric but close to symmetric $\Rightarrow$ Deformed version of a permutation of itself

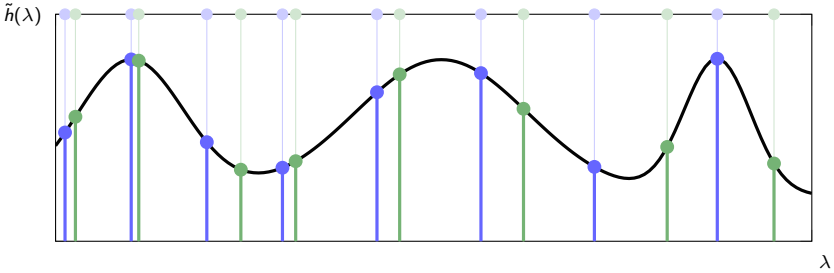


- Quasi-Symmetry, not symmetry $\Rightarrow$ Stability to deformations that are close to permutation.
- GNNs have better stability properties than graph filters $\Rightarrow$ Better at leveraging quasi-symmetries.

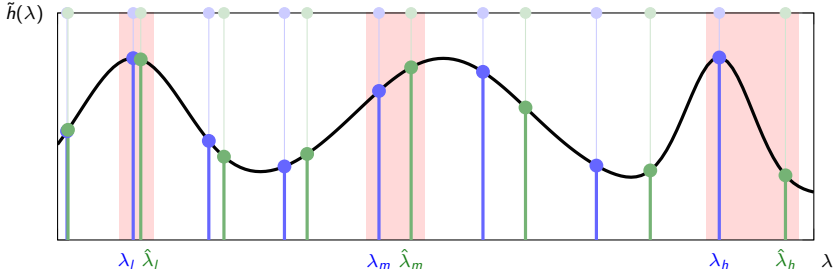
- Graph filters are operators defined on graph shift operators $\Rightarrow \mathbf{H}(\mathbf{S}) = \sum_{k=1}^{\infty} h_k \mathbf{S}^k = \mathbf{V} \sum_{k=1}^{\infty} h_k \mathbf{\Lambda}^k \mathbf{V}^H$
- They are completely characterized by their frequency responses $\Rightarrow \tilde{h}(\lambda) = \sum_{k=1}^{\infty} h_k \lambda^k$



- Graph $\mathbf{S}$ has eigenvalues λ_i $\Rightarrow$ The response is instantiated at these eigenvalues $\tilde{h}(\lambda_i) = \sum_{k=1}^{\infty} h_k \lambda_i^k$
- Graph $\hat{\mathbf{S}}$ has eigenvalues $\hat{\lambda}_i$ $\Rightarrow$ The response is instantiated at these eigenvalues $\tilde{h}(\hat{\lambda}_i) = \sum_{k=1}^{\infty} h_k \hat{\lambda}_i^k$



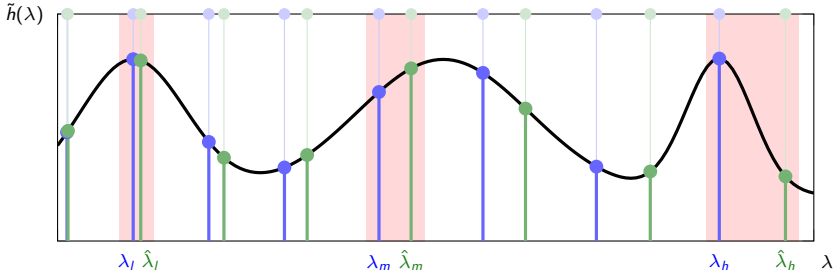
- Meaningful perturbations of a shift operator operator are relative $\Rightarrow \mathbf{P}^T \hat{\mathbf{S}} \mathbf{P} = \mathbf{S} + \mathbf{ES} + \mathbf{SE}$
- Conceptually, we learn all there is to be learnt from dilations $\Rightarrow \hat{\mathbf{S}} = \mathbf{S} + \epsilon \mathbf{S}$
- Eigenvalues dilate $\lambda_i \to \hat{\lambda}_i = (1+\epsilon)\lambda_i$. Frequency response instantiated on dilated eigenvalues



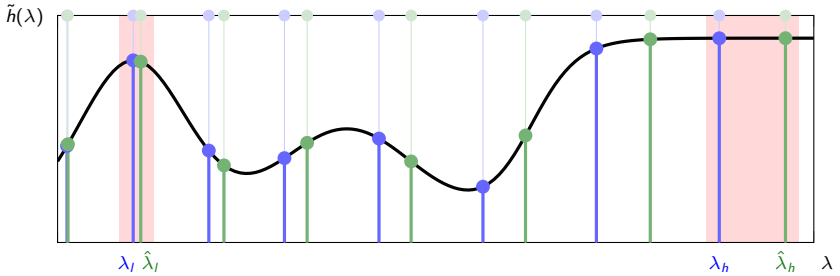
- Large eigenvalues move more. Signals with high frequencies are more difficult to process

  $\Rightarrow$ Even small perturbations yield large differences in the filter values that are instantiated

  $\Rightarrow$ We think we instantiate $h\Big(\lambda_i\Big)$ $\Rightarrow$ But in reality we instantiate $h\Big(\hat{\lambda}_i\Big) = h\Big((1+\epsilon)\,\lambda_i\Big)$

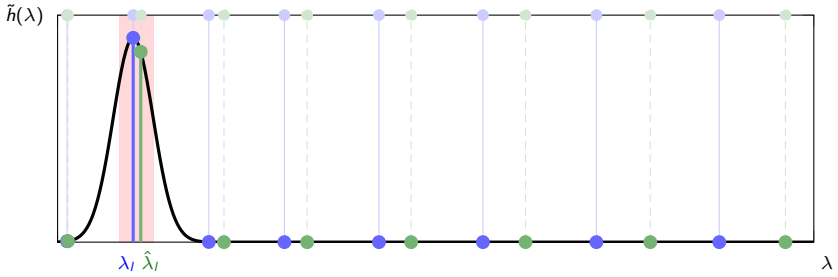


- To attain stable graph signal processing we need integral Lipschitz filters $\Rightarrow \left|\lambda \tilde{h}'(\lambda)\right| \leq C$
- Either the eigenvalue does not change because we are considering low frequencies
- Or the frequency response does not change when we are considering high frequencies



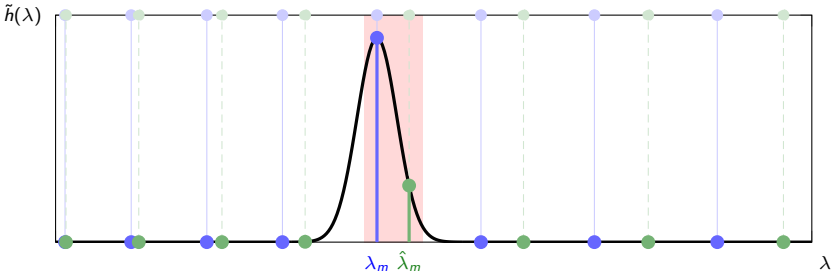
- At low frequencies a sharp highly discriminative filter is also highly stable

$\Rightarrow$ Ideal response $h\Big(\lambda_l\Big)$ is very close to perturbed response $h\Big(\hat{\lambda}_l\Big) = h\Big((1+\epsilon)\,\lambda_l\Big)$



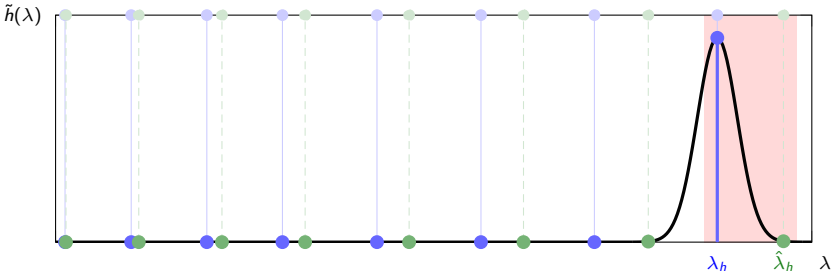
- At intermediate frequencies a sharp highly discriminative filter is somewhat stable

$\Rightarrow$ Ideal response $h\Big(\lambda_m\Big)$ is somewhat close to perturbed response $h\Big(\hat{\lambda}_m\Big) = h\Big((1+\epsilon)\,\lambda_m\Big)$



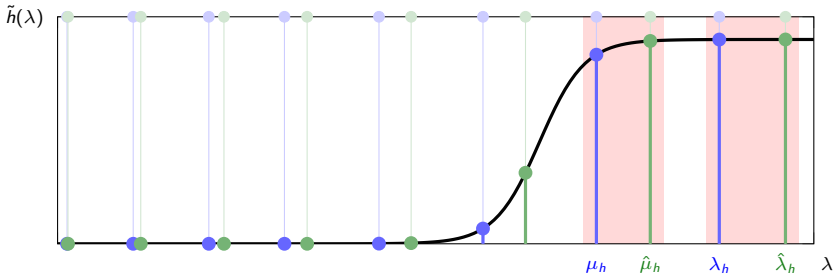
- At high frequencies a sharp highly discriminative filter is unstable. It becomes useless

$\Rightarrow$ Ideal response $h\Big(\lambda_h\Big)$ is very different from perturbed response $h\Big(\hat{\lambda}_h\Big) = h\Big((1+\epsilon)\,\lambda_h\Big)$



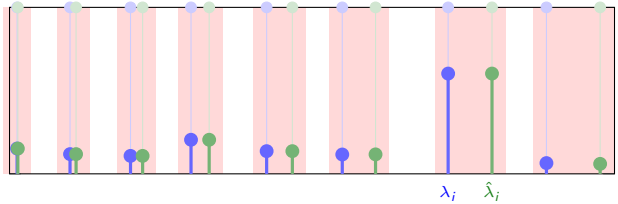
- We can have stability to deformations if we use an integral Lipschitz filters $\Rightarrow \left| \lambda \tilde{h}'(\lambda) \right| \leq C$

  $\Rightarrow$ But this precludes the discrimination of high frequency components

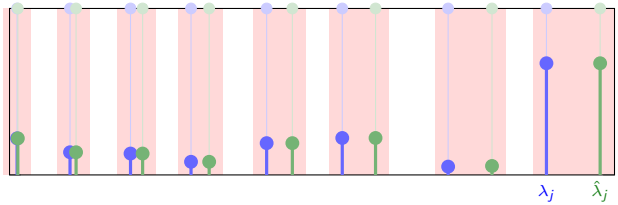


- Nonlinearities $\sigma(\mathbf{v}_i)$ and $\sigma(\mathbf{v}_j)$ spread energy across all frequencies
- Some energy where it used to be
- Some energy at low frequencies
- Where it can be discriminated with a stable filter in Layer 2

Spectrum of nonlinearity applied to $\mathbf{v}_i \Rightarrow \mathbf{V}^H\sigma(\mathbf{v}_i)$



Spectrum of nonlinearity applied to $\mathbf{v}_j \Rightarrow \mathbf{V}^H\sigma(\mathbf{v}_j)$



Fact 2: Stability Properties of GNNs

GNNs can be simultaneously discriminative and stable to deformations. Graph filters cannot.

Fact 2: Stability Properties of GNNs

For the same sensitivity to deformations, GNNs are more discriminative than graph filters

Theorem (GNN Stability to Relative Perturbations)

Consider a GNN operator $\mathbf{\Phi}(\cdot; \mathbf{S}, \mathbf{A})$ along with shifts operators $\mathbf{S}$ and $\hat{\mathbf{S}}$ having n nodes. If:

- **(H1)** Shift operators are related by $\mathbf{P}^T \hat{\mathbf{S}} \mathbf{P} = \mathbf{S} + \mathbf{E}\mathbf{S} + \mathbf{S}\mathbf{E}$ with $\mathbf{P}$ a permutation matrix
- **(H2)** The error matrix $\mathbf{E}$ has norm $\|\mathbf{E}\| = \epsilon$ and eigenvector misalignement δ relative to $\mathbf{S}$
- **(H3)** The GNN has L single-feature layers with integral Lipschitz filters with constant C
- **(H4)** Filters have unit operator norm and the nonlinearity is normalized Lipschitz

The operator distance modulo permutation between $\mathbf{\Phi}(\cdot; \mathbf{S}, \mathbf{A})$ and $\mathbf{\Phi}(\cdot; \hat{\mathbf{S}}, \mathbf{A})$ is bounded by

$$\left\| \mathbf{\Phi}(\cdot; \hat{\mathbf{S}}, \mathbf{A}) - \mathbf{\Phi}(\cdot; \mathbf{S}, \mathbf{A}) \right\|_{\mathcal{P}} \leq 2C \left(1 + \delta\sqrt{n}\right) L\epsilon + \mathcal{O}(\epsilon^2).$$

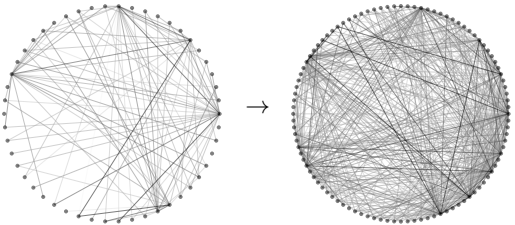
Gama-Bruna-Ribeiro, *Stability Properties of Graph Neural Networks*, TSP 2020, arxiv.org/abs/1905.04497

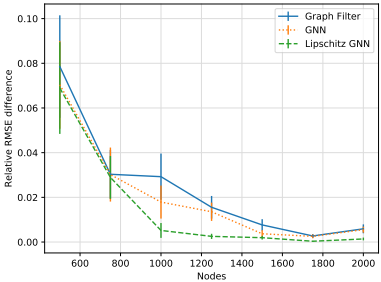
Transferability Properties of Graph Neural Networks

- A GNN that is trained in a graph $\mathbf{S}$ can be executed on any other graph $\hat{\mathbf{S}}$

  $\Rightarrow$ In particular, we can execute it in a much larger graph

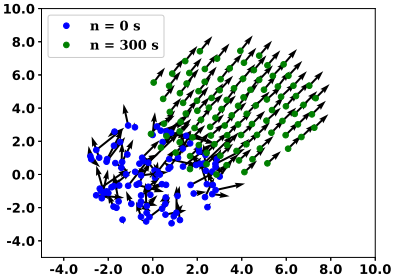
- Transferability of graph neural networks is ready to verify in practice $\Rightarrow$ recommendation system

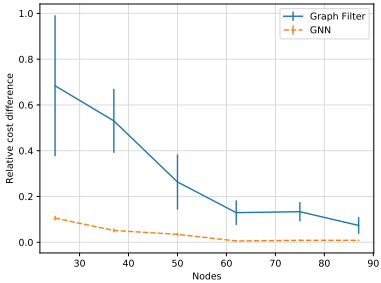




- Performance difference on training and target graphs decreases as size of training graph grows
- GNNs appear to be more transferable than graph convolutional filters $\Rightarrow$ better ML model

- Transferability of graph neural networks is ready to verify in practice $\Rightarrow$ decentralized robot control





- Performance difference on training and target graphs decreases as size of training graph grows
- GNNs appear to be more transferable than graph convolutional filters $\Rightarrow$ better ML model

Q1: We have empirically observed that GNNs transfer at scale. Why do they?

Q2: Can success of GNNs on moderate-size graphs be used to create success at large-scale?

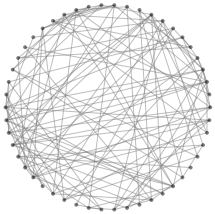
- To answer these questions, turn to CNNs $\Rightarrow$ known to scale well for images and time sequences

- Discrete time/image signals converge to continuous time/image signals $\Rightarrow \downarrow$ intrinsic dimension

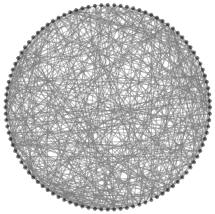
$\Rightarrow$ From SP theory, CNNs have well-defined limits on the limits of images and time signals

- A1: Intrinsic dimensionality of the problem is less than the size of the image
- A2: Training with small images is sufficient $\Rightarrow$ CIFAR 10 images are 32×32

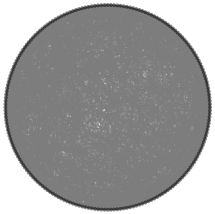
- Graphs also have limit objects that effectively limit their dimensionality $\Rightarrow$ one is the graphon



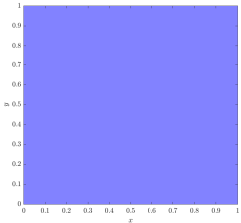
$n = 50$ nodes $\rightarrow$



$n = 100$ nodes $\rightarrow$



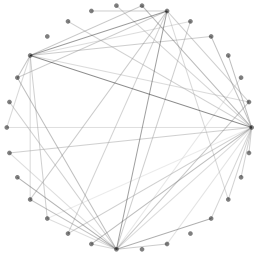
$n = 200$ nodes $\rightarrow$



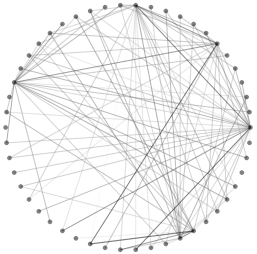
Graphon $W(u, v) = p$

- A graphon can be thought of as a graph with an uncountable number of nodes

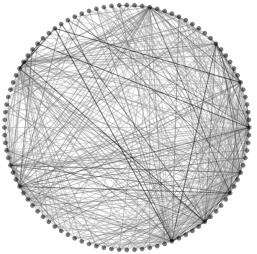
- Graphs however do not have the Euclidean structure time and image signals have in the limit



$n = 30$ products



$n = 50$ products



$n = 100$ products

- So do graph convolutions and graph neural networks converge to limits on the graphon?

Q1: We have empirically observed that GNNs scale. Why do they scale?

- A1: Because graph convolutions and GNNs have well-defined limits on graphons

L. Ruiz et al, *Graphon Signal Processing*, TSP 2021, https://arxiv.org/abs/2003.05030

L. Ruiz et al, *Transferability Properties of Graph Neural Networks*, https://arxiv.org/abs/2112.04629

Q2: Can success of GNNs on moderate-size graphs be used to create success at large-scale?

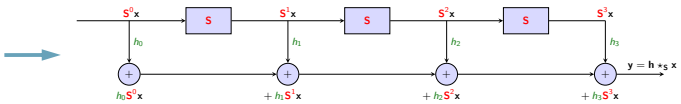
- A2: Yes, as GNNs are transferable $\Rightarrow$ can be trained on moderate-size and executed on large-scale

J. Cerviño et al, *Learning by Transference: Training Graph Neural Networks on Growing Graphs.*, https://arxiv.org/abs/2106.03693

Graphon convolutional filters and graph convolutional filters are the same algebraic object. Which is also the same algebraic object of a standard convolutional filter.

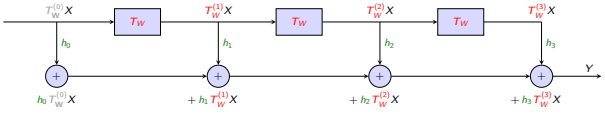
Graph convolutional filters are polynomials on a matrix representation of the graph acting on input signal.

The coefficients of the filter are the coefficients of the polynomial.



Graphon convolutional filters are polynomials on the graphon integral operator acting on input signal.

The coefficients of the filter are the coefficients of the polynomial.



Graphon integral operator: $T_W X \;:\; (T_W)X(v) \;=\; \int_0^1 \mathbf{W}(u,v)\,X(u)\,du$

WNNs are compositions of layers. Themselves compositions of graphon filters with pointwise nonlinearities

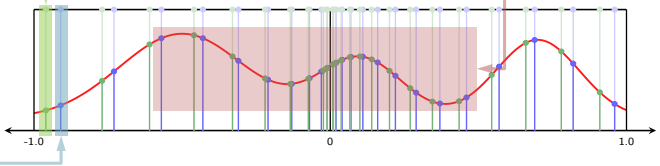
[Ruiz et al '20] Graphon Signal Processing, https://arxiv.org/abs/2003.05030

Graphon filters admit a frequency representation. Same as graph filters. Same as standard convolutions

They are still the same algebraic object: They are polynomials of scalar variables

Representation of graph filter is instantiated at graph eigenvalues

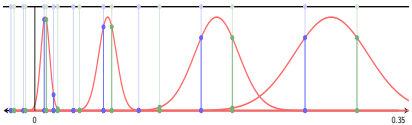
Representation of graphon filter is instantiated at graphon eigenvalues



Since graph eigenvalues converge to graphon eigenvalues convergence of graph to graphon filters follows.

The catch is that we have accumulation of eigenvalues around zero.

Thus, we can't transfer filters that attempt to discriminate these eigenvalues. There is a transferability vs discriminability tradeoff



[Ruiz et al '21] Transferability Properties of Graph Neural Networks, https://arxiv.org/abs/2112.04629

We derive a finite sample transferability bound from a graph with m nodes to a graph with n nodes

Transferability of a filter depends on the Lipschitz constant of the frequency response of the graph (and graphon) filter

Theorem (Graph Filter Transferability)

Consider graph signals $(\mathbf{S}_n, \mathbf{x}_n)$ and $(\mathbf{S}_m, \mathbf{x}_m)$ sampled from graphon signal (W, X) along with convolution outputs $\mathbf{y}_n = \mathbf{H}(\mathbf{S}_n)\mathbf{x}_n$ and $\mathbf{y}_m = \mathbf{H}(\mathbf{S}_m)\mathbf{x}_m$. The difference norm of the respective graphon induced signals is bounded by

$$\|\mathbf{Y}_n - \mathbf{Y}_m\| \leq 2A_w \left(A_h + \pi \frac{\max(B_{nc}, B_{mc})}{\min(\delta_{nc}, \delta_{mc})} \right) \left(\frac{1}{n} + \frac{1}{m} \right) \|X\| + A_x (A_h c + 2) \left(\frac{1}{n} + \frac{1}{m} \right) + 4 A_h c \|X\|$$

Same bound holds for GNNs because the pointwise nonlinearity transfers verbatim because it does not mix components

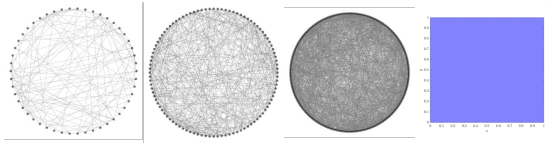
[Ruiz et al '20] Graphon Neural Networks and the Transferability of Graph Neural Networks, https://papers.nips.cc/paper/2020/hash/12bcd658ef0a540cabc36cdf2b1046fd-Abstract.html

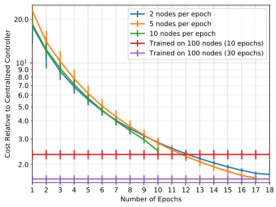
[Ruiz et al '21] Transferability Properties of Graph Neural Networks, https://arxiv.org/abs/2112.04629

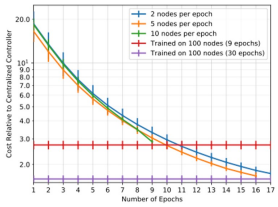
Transferability can be leveraged to learn in a sequence of growing graphs. We say that we learn by transference.

We consider graphs of growing sizes and use the GNN trained on a smaller graph as a warm start to learn the optimal GNN for a larger graph.

Faster training. Enables training in large scale graphs.

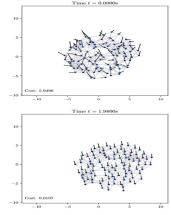






Training with growing graphs learns GNNs with the same performance

Computational cost is reduced by a 5.67 factor. More possible if graph is larger



[Cerviño et al '21] Learning by Transference: Training Graph Neural Networks on Growing Graphs, https://arxiv.org/abs/2106.03693