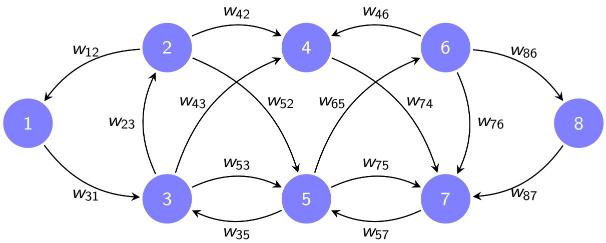
Graphs and Shift Operators

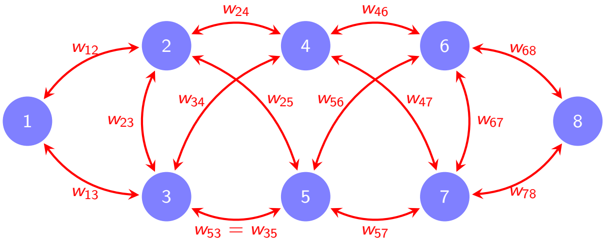
- A graph is a triplet $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, which includes vertices $\mathcal{V}$, edges $\mathcal{E}$, and weights $\mathcal{W}$
  - $\Rightarrow$ Vertices or nodes are a set of n labels. Typical labels are $\mathcal{V} = \{1, \ldots, n\}$
  - $\Rightarrow$ Edges are ordered pairs of labels (i, j). We interpret $(i, j) \in \mathcal{E}$ as "i can be influenced by j."
  - $\Rightarrow$ Weights $w_{ij} \in \mathbb{R}$ are numbers associated to edges (i, j). "Strength of the influence of j on i."



- A graph is symmetric or undirected if both, the edge set and the weight are symmetric

  $\Rightarrow$ Edges come in pairs $\Rightarrow$ We have $(i, j) \in \mathcal{E}$ if and only if $(j, i) \in \mathcal{E}$

  $\Rightarrow$ Weights are symmetric $\Rightarrow$ We must have $w_{ij} = w_{ji}$ for all $(i, j) \in \mathcal{E}$

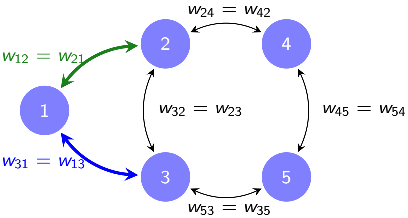


- Most of the graphs we encounter in practical situations are symmetric and weighted

- The adjacency matrix of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is the sparse matrix $\mathbf{A}$ with nonzero entries

$$A_{ij} = w_{ij}, \text{ for all } (i,j) \in \mathcal{E}$$

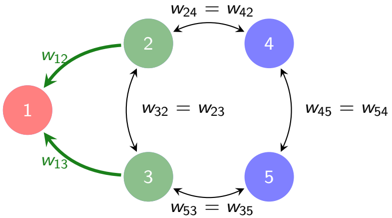
- If the graph is symmetric, the adjacency matrix is symmetric $\Rightarrow \mathbf{A} = \mathbf{A}^T$. As in the example



$$\mathbf{A} = \begin{bmatrix} 0 & w_{12} & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{23} & w_{24} & 0 \\ w_{31} & w_{32} & 0 & 0 & w_{35} \\ 0 & w_{42} & 0 & 0 & w_{45} \\ 0 & 0 & w_{53} & w_{54} & 0 \end{bmatrix}.$$

- The neighborhood of node i is the set of nodes that influence i $\Rightarrow n(i) := \{j : (i,j) \in \mathcal{E}\}$

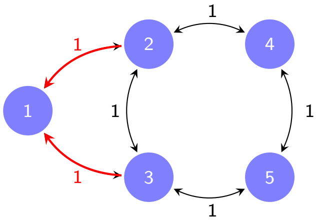
- Degree d_i of node i is the sum of the weights of its incident edges $\Rightarrow d_i = \sum_{j \in n(i)} w_{ij} = \sum_{j:(i,j)\in\mathcal{E}\}} w_{ij}$



- Node 1 neighborhood $\Rightarrow n(1) = \{2, 3\}$
- Node 1 degree $\Rightarrow n(1) = w_{12} + w_{13}$

- The degree matrix is a diagonal matrix $\mathbf{D}$ with degrees as diagonal entries $\Rightarrow D_{ii} = d_i$

- Write in terms of adjacency matrix as $\mathbf{D} = \text{diag}(\mathbf{A1})$. Because $(\mathbf{A1})_i = \sum_j w_{ij} = d_i$



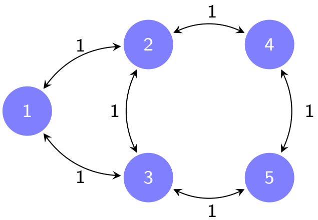
$$\mathbf{D} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

- The Laplacian matrix of a graph with adjacency matrix $\mathbf{A}$ is $\Rightarrow \mathbf{L} = \mathbf{D} - \mathbf{A} = \text{diag}(\mathbf{A1}) - \mathbf{A}$
- Can also be written explicitly in terms of graph weights $A_{ij} = w_{ij}$

$\Rightarrow$ Off diagonal entries $\Rightarrow L_{ij} = -A_{ij} = -w_{ij}$

$\Rightarrow$ Diagonal entries $\Rightarrow L_{ii} = d_i = \sum_{j \in n(i)} w_{ij}$

$$\mathbf{L} = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ -1 & -1 & 3 & 0 & -1 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & -1 & -1 & 2 \end{bmatrix}$$



- The Graph Shift Operator $\mathbf{S}$ is a stand in for any of the matrix representations of the graph

Adjacency Matrix	Laplacian Matrix	Normalized Adjacency	Normalized Laplacian
$\mathbf{S} = \mathbf{A}$	$\mathbf{S} = \mathbf{L}$	$\mathbf{S} = \bar{\mathbf{A}}$	$\mathbf{S} = \bar{\mathbf{L}}$

- If the graph is symmetric, the shift operator $\mathbf{S}$ is symmetric $\Rightarrow \mathbf{S} = \mathbf{S}^T$

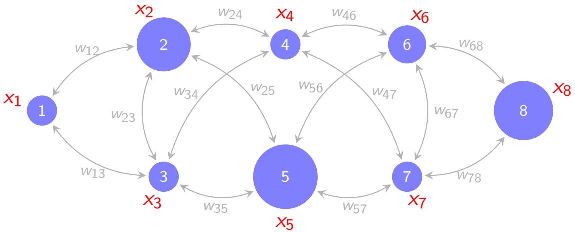
- The specific choice matters in practice but most of results and analysis hold for any choice of $\mathbf{S}$

Graph Signals

- Graph Signals are supported on a graph. They are the objets we process in Graph Signal Processing

Graph Signal

- Consider a given graph $\mathcal{G}$ with n nodes and shift operator $\mathbf{S}$
- A graph signal is a vector $\mathbf{x} \in \mathbb{R}^n$ in which component x_i is associated with node i
- To emphasize that the graph is intrinsic to the signal we may write the signal as a pair $\Rightarrow (\mathbf{S}, \mathbf{x})$



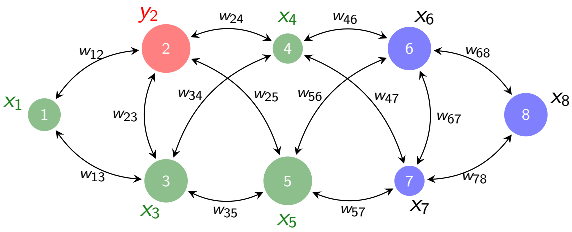
- The graph is an expectation of proximity or similarity between components of the signal $\mathbf{x}$

- Multiplication by the graph shift operator implements diffusion of the signal over the graph

- Define diffused signal $\mathbf{y} = \mathbf{S}\mathbf{x}$ $\Rightarrow$ Components are $y_i = \sum_{j \in n(i)} w_{ij}\, x_j = \sum_{j} w_{ij}\, x_j$

  $\Rightarrow$ Stronger weights contribute more to the diffusion output

  $\Rightarrow$ Codifies a local operation where components are mixed with components of neighboring nodes.



Graph Convolutional Filters

- Graph convolutional filters are the tool of choice for the linear processing of graph signals

- Given graph shift operator $\mathbf{S}$ and coefficients h_k, a graph filter is a polynomial (series) on $\mathbf{S}$

$$\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{\infty} h_k \mathbf{S}^k$$

- The result of applying the filter $\mathbf{H}(\mathbf{S})$ to the signal $\mathbf{x}$ is the signal

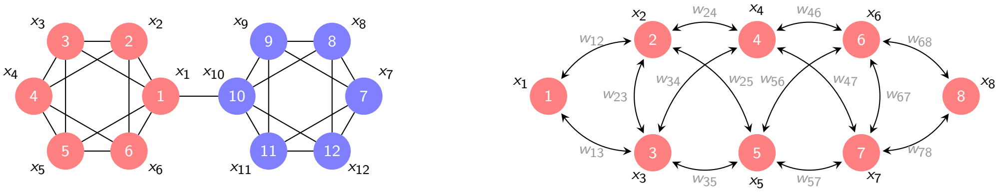
$$\mathbf{y} \;=\; \mathbf{H}(\mathbf{S})\,\mathbf{x} \;=\; \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$$

- We say that $\mathbf{y} = \mathbf{h} \star_{\mathbf{S}} \mathbf{x}$ is the graph convolution of the filter $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$ with the signal $\mathbf{x}$

- The same filter $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$ can be executed in multiple graphs $\Rightarrow$ We can transfer the filter

Graph Filter on a Graph

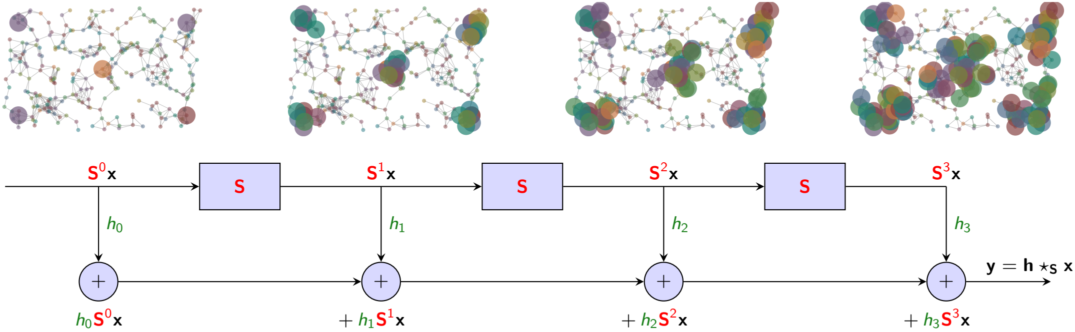
Same Graph Filter on Another Graph



- Graph convolution output $\Rightarrow \mathbf{y} = \mathbf{h} \star_{\mathbf{S}} \mathbf{x} = \; h_0\mathbf{S}^0\mathbf{x} + h_1\mathbf{S}^1\mathbf{x} + h_2\mathbf{S}^2\mathbf{x} + h_3\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\mathbf{S}^k\mathbf{x}$

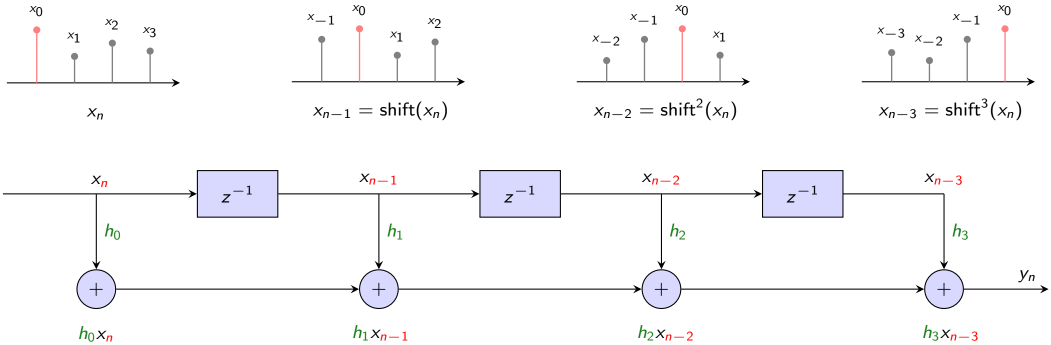
- Output depends on the filter coefficients $\mathbf{h}$, the graph shift operator $\mathbf{S}$ and the signal $\mathbf{x}$

- 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



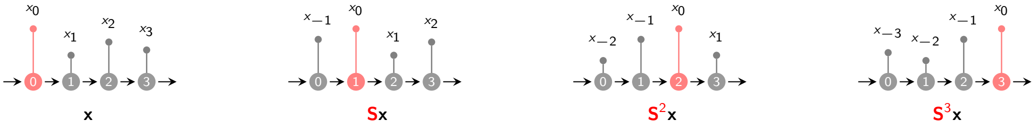
Time Convolutions as a Particular Case of Graph Convolutions

- Convolutional filters process signals in time by leveraging the time shift operator



- The time convolution is a linear combination of time shifted inputs $\Rightarrow y_n = \sum_{k=0}^{K-1} h_k x_{n-k}$

- Time signals are representable as graph signals supported on a line graph $\mathbf{S}$ $\Rightarrow$ The pair $(\mathbf{S}, \mathbf{x})$



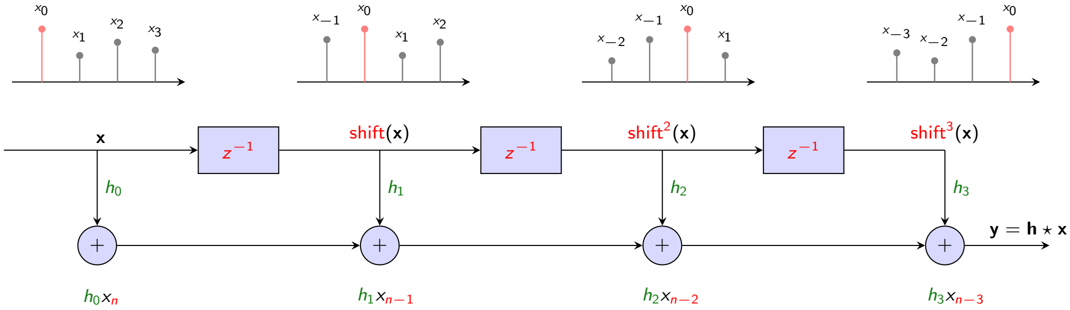
- Time shift is reinterpreted as multiplication by the adjacency matrix $\mathbf{S}$ of the line graph

$$\mathbf{S}^3\mathbf{x} = \mathbf{S}\left[\mathbf{S}^2\mathbf{x}\right] = \mathbf{S}\left[\mathbf{S}\left(\mathbf{S}\mathbf{x}\right)\right] = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & 0 & 0 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & \cdots \\ & \vdots & \vdots & \vdots & \end{bmatrix} \begin{bmatrix} \vdots \\ x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ x_{-3} \\ x_{-2} \\ x_{-1} \\ x_0 \\ \vdots \end{bmatrix}$$

- Components of the shift sequence are powers of the adjacency matrix applied to the original signal

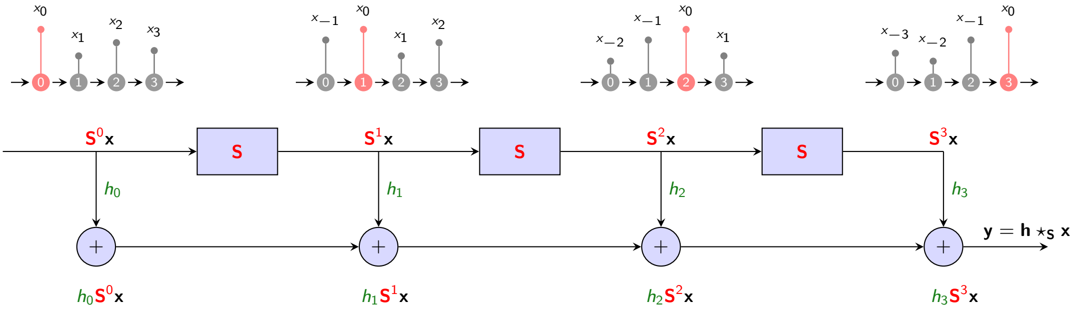
  $\Rightarrow$ We can rewrite convolutional filters as polynomials on $\mathbf{S}$, the adjacency of the line graph

- The convolution operation is a linear combination of shifted versions of the input signal
- But we now know that time shifts are multiplications with the adjacency matrix $\mathbf{S}$ of line graph



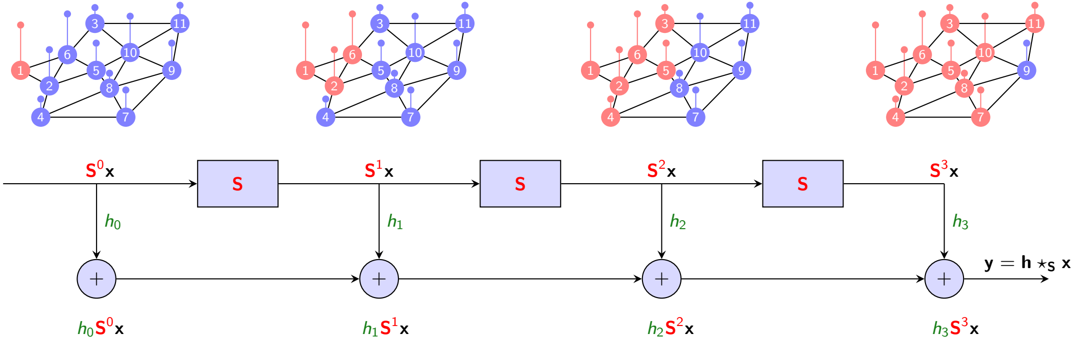
- Time convolution is a polynomial on adjacency matrix of line graph $\Rightarrow \mathbf{y} = \mathbf{h} \star \mathbf{x} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$

- The convolution operation is a linear combination of shifted versions of the input signal
- But we now know that time shifts are multiplications with the adjacency matrix $\mathbf{S}$ of line graph



- Time convolution is a polynomial on adjacency matrix of line graph $\Rightarrow \mathbf{y} = \mathbf{h} \star \mathbf{x} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$

- If we let $\mathbf{S}$ be the shift operator of an arbitrary graph we recover the graph convolution



Graph Fourier Transform

- The Graph Fourier Transform (GFT) is a tool for analyzing graph information processing systems

Eigenvectors and Eigenvalues of Shift Operator

- We work with symmetric graph shift operators $\Rightarrow \mathbf{S} = \mathbf{S}^H$

- Introduce eigenvectors $\mathbf{v}_i$ and eigenvalues λ_i of graph shift operator $\mathbf{S}$ $\Rightarrow \mathbf{S}\mathbf{v}_i = \lambda_i \mathbf{v}_i$

  $\Rightarrow$ For symmetric $\mathbf{S}$ eigenvalues are real. We have ordered them $\Rightarrow \lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_n$

- Define eigenvector matrix $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ and eigenvalue matrix $\mathbf{\Lambda} = \text{diag}([\lambda_1; \ldots; \lambda_n])$

  $\Rightarrow$ Eigenvector decomposition of Graph Shift Operator $\Rightarrow \mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$. With $\mathbf{V}^H\mathbf{V} = \mathbf{I}$

Graph Fourier Transform

Given a graph shift operator $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$, the graph Fourier transform (GFT) of graph signal $\mathbf{x}$ is

$$\tilde{\mathbf{x}} = \mathbf{V}^H \mathbf{x}$$

- The GFT is a projection on the eigenspace of the graph shift operator.
- We say $\tilde{\mathbf{x}}$ is a graph frequency representation of $\mathbf{x}$. A representation in the graph frequency domain

Inverse Graph Fourier Transform

Given a graph shift operator $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$, the inverse graph Fourier transform (iGFT) of GFT $\tilde{\mathbf{x}}$ is

$$\tilde{\tilde{\mathbf{x}}} = \mathbf{V}\,\tilde{\mathbf{x}}$$

- Given that $\mathbf{V}^H\mathbf{V} = \mathbf{I}$, the iGFT of the GFT of signal $\mathbf{x}$ recovers the signal $\mathbf{x}$

$$\tilde{\tilde{\mathbf{x}}} = \mathbf{V}\,\tilde{\mathbf{x}} = \mathbf{V}\left(\mathbf{V}^H\,\mathbf{x}\right) = \mathbf{I}\mathbf{x} = \mathbf{x}$$

Graph Frequency Response of Graph Filters

- Graph filters admit a pointwise representation when projected into the shift operator's eigenspace

Theorem (Graph frequency representation of graph filters)

Consider graph filter $\mathbf{h}$ with coefficients h_k, graph signal $\mathbf{x}$ and the filtered signal $\mathbf{y} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$.

The GFTs $\tilde{\mathbf{x}} = \mathbf{V}^H \mathbf{x}$ and $\tilde{\mathbf{y}} = \mathbf{V}^H \mathbf{y}$ are related by

$$\tilde{\mathbf{y}} = \sum_{k=0}^{\infty} h_k \mathbf{\Lambda}^k \tilde{\mathbf{x}}$$

- The same polynomial but on different variables. One on $\mathbf{S}$. The other on eigenvalue matrix $\mathbf{\Lambda}$

- In the graph frequency domain graph filters are a diagonal matrices $\Rightarrow \tilde{\mathbf{y}} = \sum_{k=0}^{\infty} h_k \mathbf{\Lambda}^k \tilde{\mathbf{x}}$

- Thus, graph convolutions are pointwise in the GFT domain $\Rightarrow \tilde{y}_i = \sum_{k=0}^{\infty} h_k \lambda_i^k \tilde{x}_i = \tilde{h}(\lambda_i)\tilde{x}_i$

Definition (Frequency Response of a Graph Filter)

Given a graph filter with coefficients $\mathbf{h} = \{h_k\}_{k=1}^{\infty}$, the graph frequency response is the polynomial

$$\tilde{h}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$$

Definition (Frequency Response of a Graph Filter)

Given a graph filter with coefficients $\mathbf{h} = \{h_k\}_{k=1}^{\infty}$, the graph frequency response is the polynomial

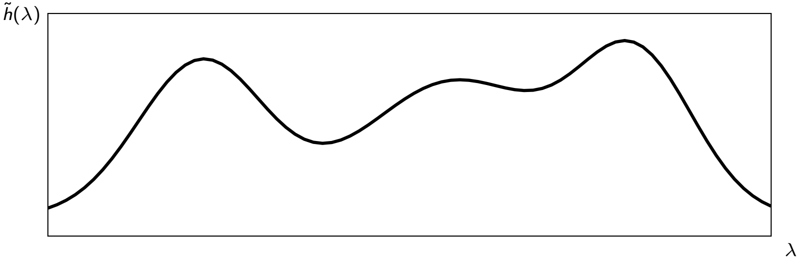
$$\tilde{h}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$$

- Frequency response is the same polynomial that defines the graph filter $\Rightarrow$ but on scalar variable λ
- Frequency response is independent of the graph $\Rightarrow$ Depends only on filter coefficients
- The role of the graph is to determine the eigenvalues on which the response is instantiated

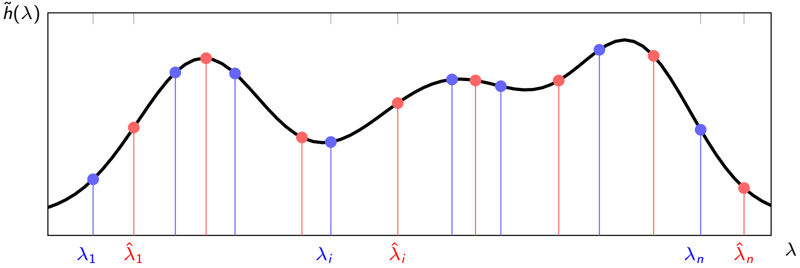
Graph Frequency Response is Independent of the Graph

- Graph filter frequency response is a polynomial on a scalar variable λ $\Rightarrow$ $\tilde{h}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$

- Completely determined by the filter coefficients $\mathbf{h} = \{h_k\}_{k=1}^{\infty}$. The Graph has nothing to do with it



- A given (another) graph instantiates the response on its given (different) specific eigenvalues λ_i
- Eigenvectors do not appear in the frequency response. They determine the meaning of frequencies.



Learning with Graph Signals

- Almost ready to introduce GNNs. We begin with a short discussion of learning with graph signals

- In this course, machine learning (ML) on graphs $\equiv$ empirical risk minimization (ERM) on graphs.
- In ERM we are given:

  $\Rightarrow$ A training set $\mathcal{T}$ containing observation pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}$. Assume equal length $\mathbf{x}, \mathbf{y}, \in \mathbb{R}^n$.

  $\Rightarrow$ A loss function $\ell(\mathbf{y}, \hat{\mathbf{y}})$ to evaluate the similarity between $\mathbf{y}$ and an estimate $\hat{\mathbf{y}}$

  $\Rightarrow$ A function class $\mathcal{C}$

- Learning means finding function $\Phi^* \in \mathcal{C}$ that minimizes loss $\ell\Big(\mathbf{y}, \Phi(\mathbf{x})\Big)$ averaged over training set

$$\Phi^* = \underset{\Phi \in \mathcal{C}}{\text{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{y}, \Phi(\mathbf{x}),\Big)$$

- We use $\Phi^*(\mathbf{x})$ to estimate outputs $\hat{\mathbf{y}} = \Phi^*(\mathbf{x})$ when inputs $\mathbf{x}$ are observed but outputs $\mathbf{y}$ are unknown

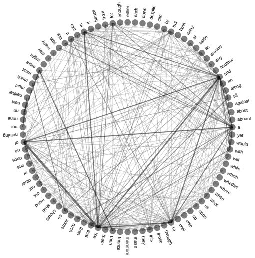
Empirical Risk Minimization with Graph Signals

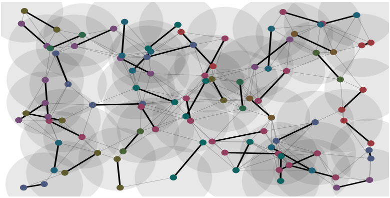
- In ERM, the function class $\mathcal{C}$ is the degree of freedom available to the system's designer

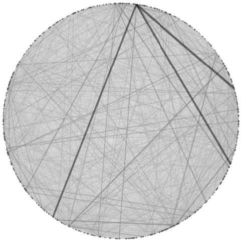
$$\Phi^* = \underset{\Phi \in \mathcal{C}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{y}, \Phi(\mathbf{x})\Big)$$

- Designing a Machine Learning $\equiv$ finding the right function class $\mathcal{C}$

- Since we are interested in graph signals, graph convolutional filters are a good starting point







- Input / output signals $\mathbf{x}$ / $\mathbf{y}$ are graph signals supported on a common graph with shift operator $\mathbf{S}$

- Function class $\Rightarrow$ graph filters of order K supported on $\mathbf{S}$ $\Rightarrow$ $\Phi(\mathbf{x}) = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x} = \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h})$

$$\mathbf{x} \longrightarrow \boxed{\mathbf{z} = \sum_{k=0}^{K-1} h_k \, \mathbf{S}^k \, \mathbf{x}} \longrightarrow \mathbf{z} = \boldsymbol{\Phi}(\mathbf{x}; \mathbf{S}, \mathbf{h})$$

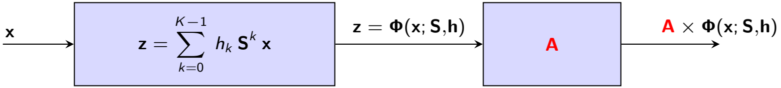
- Learn ERM solution restricted to graph filter class $\Rightarrow$ $\mathbf{h}^* = \underset{\mathbf{h}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{y}, \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h})\Big)$

$\Rightarrow$ Optimization is over filter coefficients $\mathbf{h}$ with the graph shift operator $\mathbf{S}$ given

When the Output is Not a Graph Signal: Readout

- Outputs $\mathbf{y} \in \mathbb{R}^m$ are not graph signals $\Rightarrow$ Add readout layer at filter's output to match dimensions

- Readout matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ yields parametrization $\Rightarrow$ $\mathbf{A} \times \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h}) = \mathbf{A} \times \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$



- Making $\mathbf{A}$ trainable is inadvisable. Learn filter only. $\Rightarrow \mathbf{h}^* = \underset{\mathbf{h}}{\operatorname{argmin}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{y}, \mathbf{A} \times \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h})\Big)$

- Readouts are simple. Read out node i $\Rightarrow \mathbf{A} = \mathbf{e}_i^T$. Read out signal average $\Rightarrow \mathbf{A} = \mathbf{1}^T$.

Graph Neural Networks (GNNs)

[1] F. Gama, et.al, "Convolutional Neural Network Architectures for Signals Supported on Graphs," IEEE-TSP. Arxiv: 1805.00165
[2] F. Gama, et.al, "Graphs, Convolutions, and Neural Networks: From Graph Filters to Graph Neural Networks," IEEE-SPM. Arxiv: 2003.03777

- A pointwise nonlinearity is a nonlinear function applied componentwise. Without mixing entries

- The result of applying pointwise σ to a vector $\mathbf{x}$ is $\Rightarrow \sigma\left[\mathbf{x}\right] = \sigma\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_n) \end{bmatrix}$

- A pointwise nonlinearity is the simplest nonlinear function we can apply to a vector

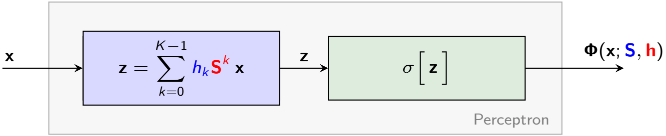
- ReLU: $\sigma(x) = \max(0, x)$. Hyperbolic tangent: $\sigma(x) = (e^{2x} - 1)/(e^{2x} + 1)$. Absolute value: $\sigma(x) = |x|$.

- Pointwise nonlinearities decrease variability. $\Rightarrow$ They function as demodulators.

Learning with a Graph Perceptron

- Graph filters have limited expressive power because they can only learn linear maps

- A first approach to nonlinear maps is the graph perceptron $\Rightarrow \Phi(\mathbf{x}) = \sigma\left[\sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}\right] = \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h})$



$$\sigma\left[\mathbf{x}\right] = \sigma\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_n) \end{bmatrix}$$

- Optimal regressor restricted to perceptron class $\Rightarrow \mathbf{h}^* = \underset{\mathbf{h}}{\text{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{y},\ \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h})\Big)$

$\Rightarrow$ Perceptron allows learning of nonlinear maps $\Rightarrow$ More expressive. Larger Representable Class

- To define a GNN we compose several graph perceptrons $\Rightarrow$ We layer graph perceptrons

- Layer 1 processes input signal $\mathbf{x}$ with the perceptron $\mathbf{h}_1 = [h_{10}, \ldots, h_{1,K-1}]$ to produce output $\mathbf{x}_1$

$$\mathbf{x}_1 = \sigma\Big[\mathbf{z}_1\Big] = \sigma\left[\sum_{k=0}^{K-1} h_{1k}\, \mathbf{S}^k\, \mathbf{x}\right]$$

- The Output of Layer 1 $\mathbf{x}_1$ becomes an input to Layer 2. Still $\mathbf{x}_1$ but with different interpretation

- Repeat analogous operations for L times (the GNNs depth) $\Rightarrow$ Yields the GNN predicted output $\mathbf{x}_L$

- To define a GNN we compose several graph perceptrons $\Rightarrow$ We layer graph perceptrons

- Layer 2 processes its input signal $\mathbf{x}_1$ with the perceptron $\mathbf{h}_2 = [h_{20}, \ldots, h_{2,K-1}]$ to produce output $\mathbf{x}_2$

$$\mathbf{x}_2 = \sigma\Big[\,\mathbf{z}_2\,\Big] = \sigma\left[\sum_{k=0}^{K-1} h_{2k}\, \mathbf{S}^k\, \mathbf{x}_1\right]$$

- The Output of Layer 2 $\mathbf{x}_2$ becomes an input to Layer 3. Still $\mathbf{x}_2$ but with different interpretation

- Repeat analogous operations for L times (the GNNs depth) $\Rightarrow$ Yields the GNN predicted output $\mathbf{x}_L$

The GNN Layer Recursion

- A generic layer of the GNN, Layer ℓ, takes as input the output $\mathbf{x}_{\ell-1}$ of the previous layer $(\ell-1)$

- Layer ℓ processes its input signal $\mathbf{x}_{\ell-1}$ with perceptron $\mathbf{h}_\ell = [h_{\ell 0}, \ldots, h_{\ell,K-1}]$ to produce output $\mathbf{x}_\ell$

$$\mathbf{x}_\ell = \sigma\Big[\mathbf{z}_\ell\Big] = \sigma\left[\sum_{k=0}^{K-1} h_{\ell k}\, \mathbf{S}^k\, \mathbf{x}_{\ell-1}\right]$$

- With the convention that the Layer 1 input is $\mathbf{x}_0 = \mathbf{x}$, this provides a recursive definition of a GNN

- If it has L layers, the GNN output $\Rightarrow \mathbf{x}_L = \Phi\Big(\mathbf{x};\, \mathbf{S},\, \mathbf{h}_1, \ldots, \mathbf{h}_L\Big) = \Phi\Big(\mathbf{x};\, \mathbf{S},\, \mathcal{H}\Big)$

- The filter tensor $\mathcal{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_L]$ is the trainable parameter. The graph shift is prior information

GNN Block Diagram

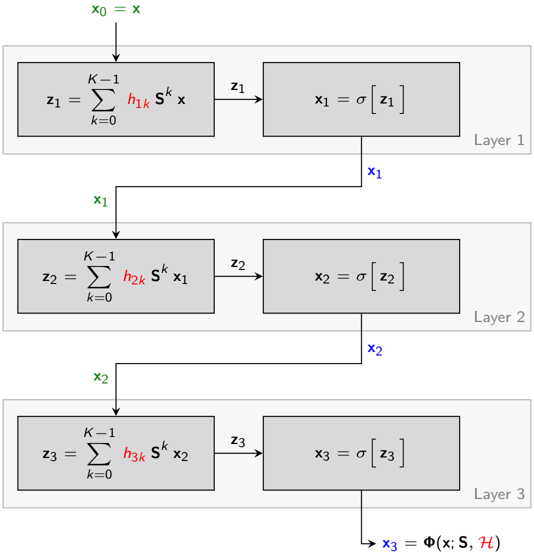
- Illustrate definition with a GNN with 3 layers

- Feed input signal $\mathbf{x} = \mathbf{x}_0$ into Layer 1

$$\mathbf{x}_1 = \sigma\Big[\mathbf{z}_1\Big] = \sigma\left[\sum_{k=0}^{K-1} h_{1k}\,\mathbf{S}^k\,\mathbf{x}_0\right]$$

- Last layer output is the GNN output $\Rightarrow \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

$\Rightarrow$ Parametrized by filter tensor $\mathcal{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$



GNN Block Diagram

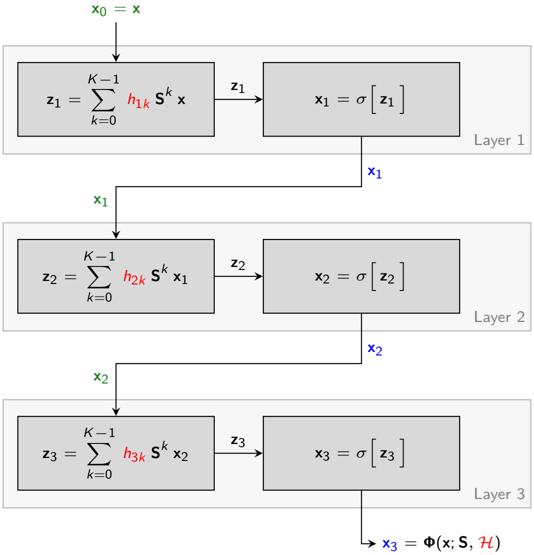
- Illustrate definition with a GNN with 3 layers

- Feed Layer 1 output as an input to Layer 2

$$\mathbf{x}_2 = \sigma\Big[\,\mathbf{z}_2\,\Big] = \sigma\left[\sum_{k=0}^{K-1} h_{2k}\,\mathbf{S}^k\,\mathbf{x}_1\right]$$

- Last layer output is the GNN output $\Rightarrow \Phi(\mathbf{x};\mathbf{S},\mathcal{H})$

  $\Rightarrow$ Parametrized by filter tensor $\mathcal{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$



GNN Block Diagram

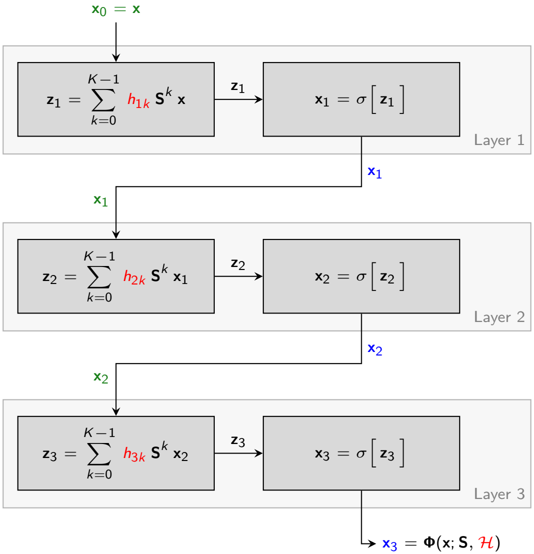
- Illustrate definition with a GNN with 3 layers

- Feed Layer 2 output as an input to Layer 3

$$\mathbf{x}_3 = \sigma\Big[\,\mathbf{z}_3\,\Big] = \sigma\left[\sum_{k=0}^{K-1} h_{3k}\, \mathbf{S}^k\, \mathbf{x}_2\right]$$

- Last layer output is the GNN output $\Rightarrow \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

  $\Rightarrow$ Parametrized by filter tensor $\mathcal{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$



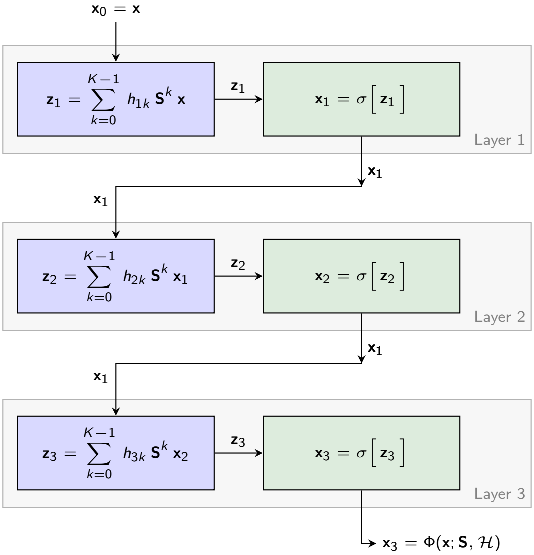
Some Observations about Graph Neural Networks

- A GNN with L layers follows L recursions of the form

$$\mathbf{x}_\ell = \sigma\Big[\,\mathbf{z}_\ell\,\Big] = \sigma\left[\sum_{k=0}^{K-1} h_{\ell k}\,\mathbf{S}^k\,\mathbf{x}_{\ell-1}\right]$$

- A composition of L layers. Each of which itself a...

$\Rightarrow$ Compositions of Filters & Pointwise nonlinearities

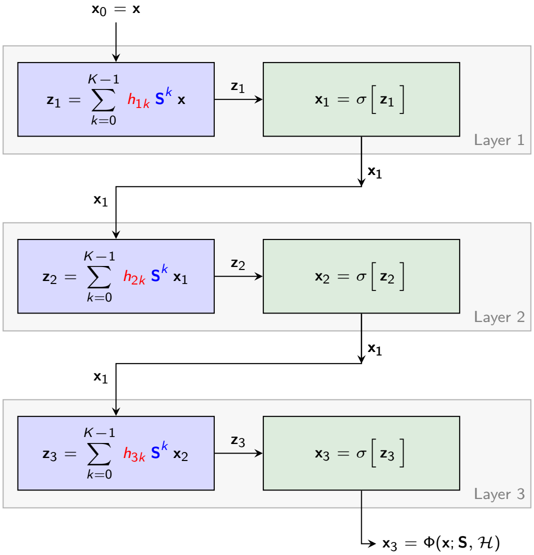


- A GNN with L layers follows L recursions of the form

$$\mathbf{x}_\ell = \sigma\Big[\,\mathbf{z}_\ell\,\Big] = \sigma\left[\,\sum_{k=0}^{K-1} h_{\ell k}\,\mathbf{S}^k\,\mathbf{x}_{\ell-1}\right]$$

- Filters are parametrized by...

$\Rightarrow$ Coefficients $h_{\ell k}$ and graph shift operators $\mathbf{S}$

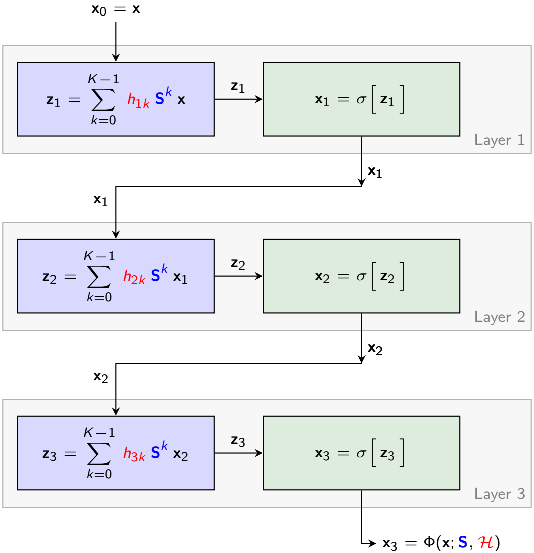


- A GNN with L layers follows L recursions of the form

$$\mathbf{x}_\ell = \sigma\Big[\,\mathbf{z}_\ell\,\Big] = \sigma\left[\sum_{k=0}^{K-1} h_{\ell k}\,\mathbf{S}^k\,\mathbf{x}_{\ell-1}\right]$$

- Output $\mathbf{x}_L = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$ parametrized by...

$\Rightarrow$ Learnable Filter tensor $\mathcal{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_L]$

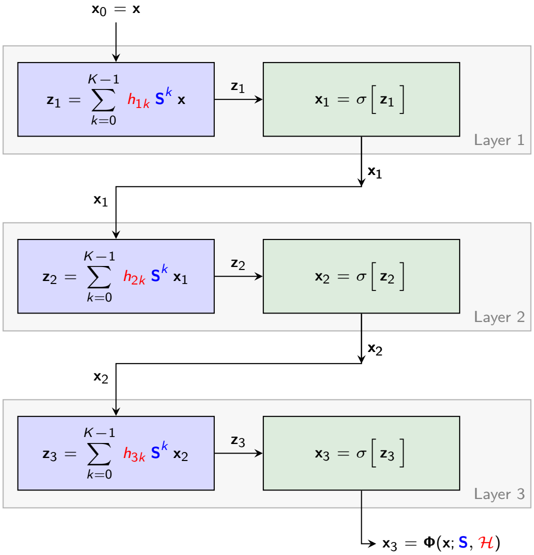


- Learn Optimal GNN tensor $\mathcal{H}^* = (\mathbf{h}_1^*, \mathbf{h}_2^*, \mathbf{h}_3^*)$ as

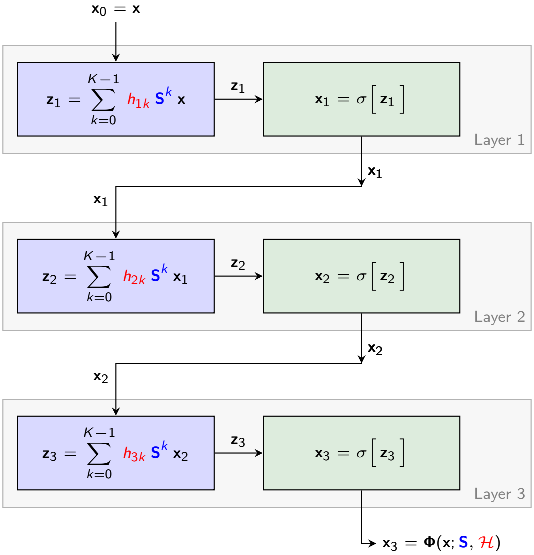
$$\mathcal{H}^* = \underset{\mathcal{H}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\boldsymbol{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}), \mathbf{y}\Big)$$

- Optimization is over tensor only. Graph $\mathbf{S}$ is given

$\Rightarrow$ Prior information given to the GNN

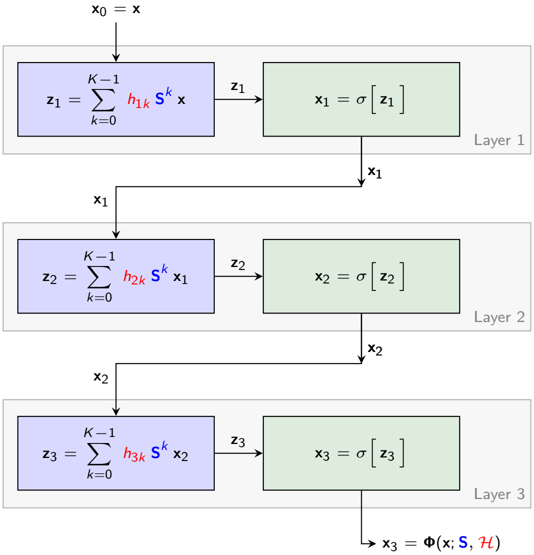


- GNNs are minor variations of graph filters
- Add pointwise nonlinearities and layer compositions
  - $\Rightarrow$ Nonlinearities process individual entries
  - $\Rightarrow$ Component mixing is done by graph filters only
- GNNs do work (much) better than graph filters
  - $\Rightarrow$ Which is unexpected and deserves explanation
  - $\Rightarrow$ Which we will attempt with stability analyses



- GNN Output depends on the graph $\mathbf{S}$.
- Interpret $\mathbf{S}$ as a parameter

  $\Rightarrow$ Encodes prior information. As we have done so far



- But we can reinterpret $\mathbf{S}$ as an input of the GNN

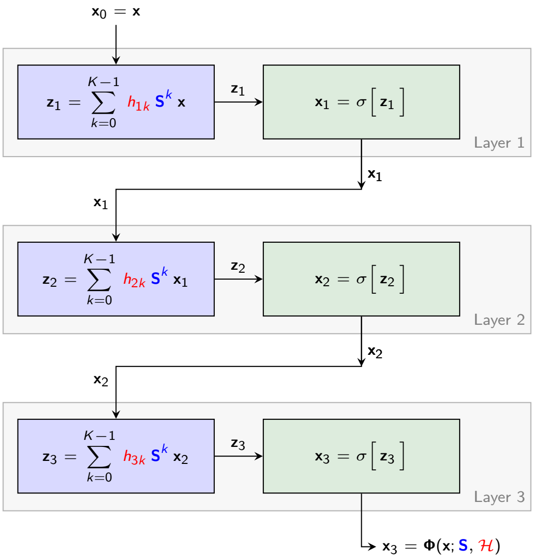
  $\Rightarrow$ Enabling transference across graphs

  $$\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}) \Rightarrow \mathbf{\Phi}(\mathbf{x}; \tilde{\mathbf{S}}, \mathcal{H})$$

  $\Rightarrow$ Same as we enable transference across signals

  $$\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}) \Rightarrow \mathbf{\Phi}(\tilde{\mathbf{x}}; \mathbf{S}, \mathcal{H})$$

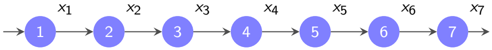
- A trained GNN is just a filter tensor $\mathcal{H}^*$



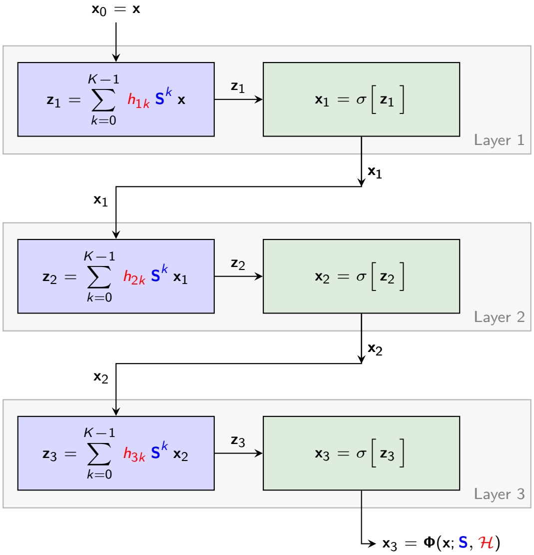
- There is no difference between CNNs and GNNs

- To recover a CNN just particularize the shift operator the adjacency matrix of the directed line graph

$$\mathbf{S} = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & 0 & 0 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & \cdots \\ & \vdots & \vdots & \vdots & \end{bmatrix}$$



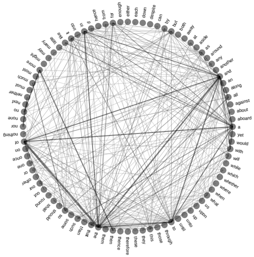
- GNNs are proper generalizations of CNNs

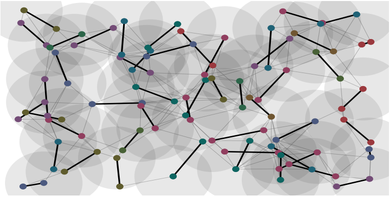


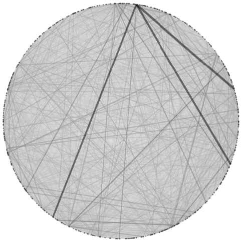
Fully Connected Neural Networks

The Road Not Taken: Fully Connected Neural Networks

- We chose graph filters and graph neural networks (GNNs) because of our interest in graph signals
- We argued this is a good idea because they are generalizations of convolutional filters and CNNs
- We can explore this better if we go back to the road not taken $\Rightarrow$ Fully connected neural networks







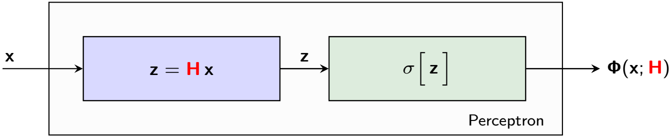
- Instead of graph filters, we choose arbitrary linear functions $\Rightarrow \boldsymbol{\Phi}(\mathbf{x}) = \boldsymbol{\Phi}(\mathbf{x}; \mathbf{H}) = \mathbf{H}\,\mathbf{x}$

$$\mathbf{x} \longrightarrow \boxed{\mathbf{z} = \mathbf{H}\,\mathbf{x}} \longrightarrow \mathbf{z} = \boldsymbol{\Phi}(\mathbf{x}; \mathbf{H})$$

- Optimal regressor is ERM solution restricted to linear class $\Rightarrow \mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\boldsymbol{\Phi}(\mathbf{x}; \mathbf{H}), \mathbf{y}\Big)$

Learning with a Linear Perceptron

- We increase expressive power with the introduction of a perceptrons $\Rightarrow \mathbf{\Phi}(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x}; \mathbf{H}) = \sigma\Big[\mathbf{H}\mathbf{x}\Big]$



- Optimal regressor restricted to perceptron class $\Rightarrow \mathbf{H}^* = \underset{\mathbf{H}}{\text{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{\Phi}(\mathbf{x}; \mathbf{H}), \mathbf{y}\Big)$

- A generic layer, Layer ℓ of a FCNN, takes as input the output $\mathbf{x}_{\ell-1}$ of the previous layer $(\ell - 1)$

- Layer ℓ processes its input signal $\mathbf{x}_{\ell-1}$ with a linear perceptron $\mathbf{H}_\ell$ to produce output $\mathbf{x}_\ell$

$$\mathbf{x}_\ell = \sigma\Big[\mathbf{z}_\ell\Big] = \sigma\Big[\mathbf{H}_\ell\,\mathbf{x}_{\ell-1}\Big]$$

- With the convention that the Layer 1 input is $\mathbf{x}_0 = \mathbf{x}$, this provides a recursive definition of a FCNN

- If it has L layers, the FCNN output $\Rightarrow \mathbf{x}_L \;=\; \Phi\Big(\mathbf{x}; \mathbf{H}_1, \ldots, \mathbf{H}_L\Big) \;=\; \Phi\Big(\mathbf{x}; \mathcal{H}\Big)$

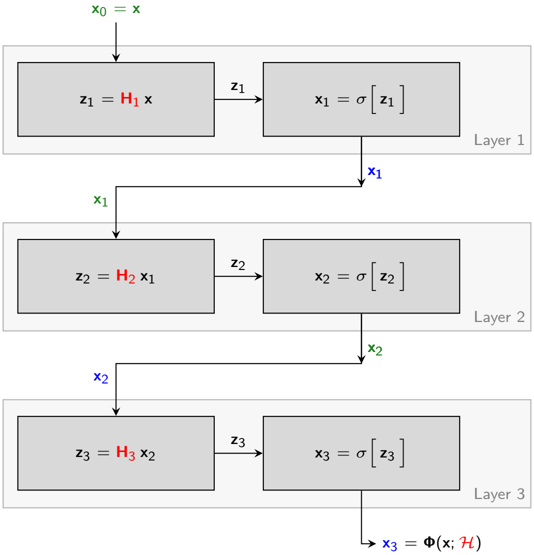
- The filter tensor $\mathcal{H} = [\mathbf{H}_1, \ldots, \mathbf{H}_L]$ is the trainable parameter.

- Illustrate definition with an FCNN with 3 layers

- Feed input signal $\mathbf{x} = \mathbf{x}_0$ into Layer 1

$$\mathbf{x}_1 = \sigma\Big[\,\mathbf{z}_1\,\Big] = \sigma\Big[\,\mathbf{H}_1\,\mathbf{x}_0\Big]$$

- Output $\Phi(\mathbf{x};\mathcal{H})$ Parametrized by $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$

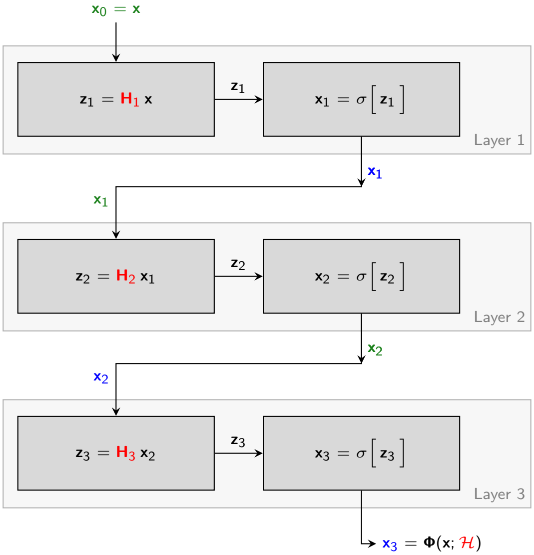


- Illustrate definition with an FCNN with 3 layers

- Feed Layer 1 output as an input to Layer 2

$$\mathbf{x}_2 = \sigma\Big[\,\mathbf{z}_2\,\Big] = \sigma\Big[\,\mathbf{H}_2\,\mathbf{x}_1\,\Big]$$

- Output $\Phi(\mathbf{x};\mathcal{H})$ Parametrized by $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$

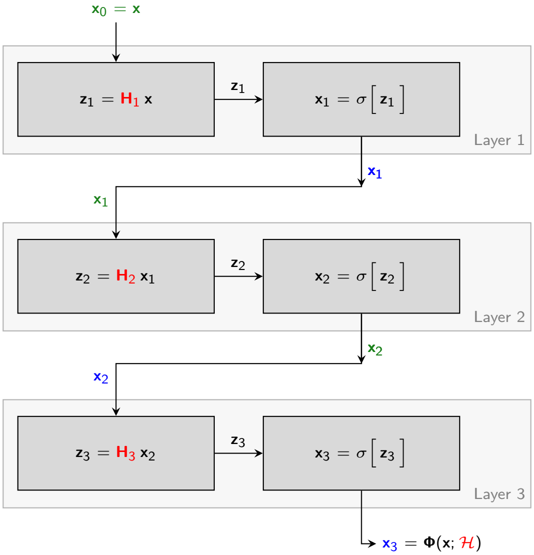


Fully Connected Neural Network Block Diagram

- Illustrate definition with an FCNN with 3 layers
- Feed Layer 2 output as an input to Layer 3

$$\mathbf{x}_3 = \sigma\Big[\,\mathbf{z}_3\,\Big] = \sigma\Big[\,\mathbf{H}_3\,\mathbf{x}_2\Big]$$

- Output $\mathbf{\Phi}(\mathbf{x}; \mathcal{H})$ Parametrized by $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$



Neural Networks vs Graph Neural Networks

Which is Better: A Graph NN or a Fully Connected NN?

- Since the GNN is a particular case of a fully connected NN, the latter attains a smaller cost

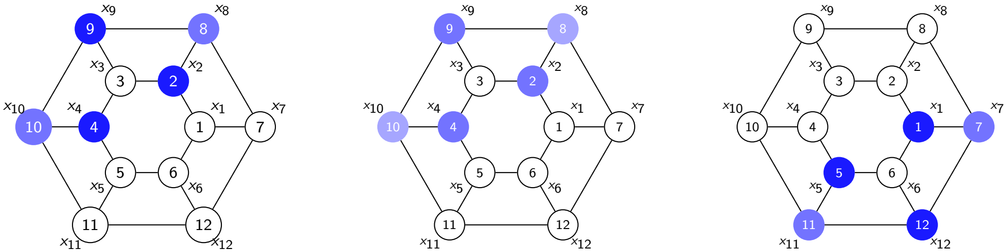
$$\min_{\mathcal{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{\Phi}(\mathbf{x};\mathcal{H}),\mathbf{y}\Big) \leq \min_{\mathcal{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H}),\mathbf{y}\Big)$$

- The fully connected NN does better. But this holds for the training set

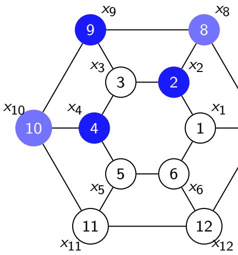
- In practice, the GNN does better because it generalizes better to unseen signals

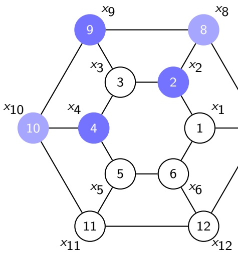
$\Rightarrow$ Because it exploits internal symmetries of graph signals codified in the graph shift operator

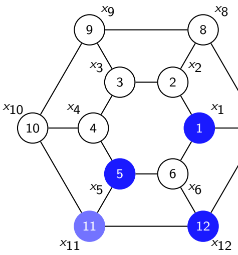
- Suppose the graph represents a recommendation system where we want to fill empty ratings
- We observe ratings with the structure in the left. But we do not observe examples like the other two
- From examples like the one in the left, the NN learns how to fill the middle signal but not the right



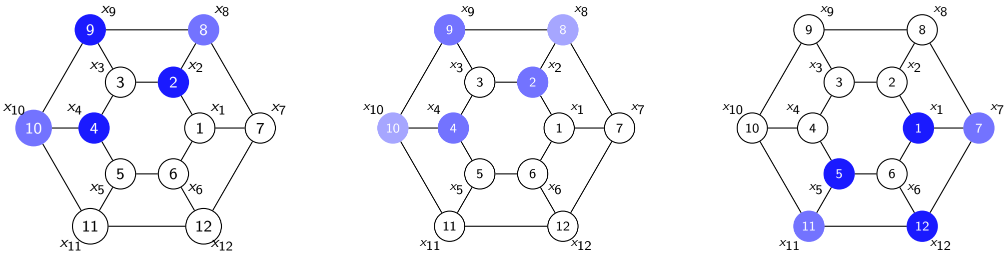
- The GNN will succeed at predicting ratings for the signal on the right because it knows the graph
- The GNN still learns how to fill the middle signal. But it also learns how to fill the right signal







- The GNN exploits symmetries of the signal to effectively multiply available data
- This will be formalized later as the permutation equivariance of graph neural networks



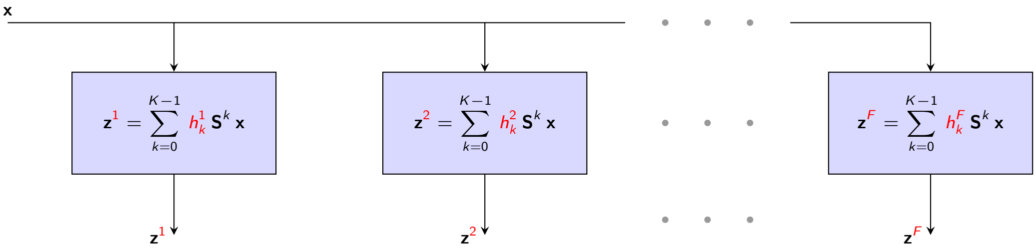
Graph Filter Banks

- Filters isolate features. When we are interested in multiple features, we use Banks of filters

Graph Filter Banks

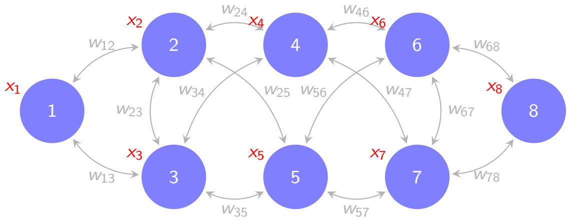
- A graph filter bank is a collection of filters. Use F to denote total number of filters in the bank

- Filter f in the bank uses coefficients $\mathbf{h}^f = [h_1^f; \ldots; h_{K-1}^f]$ $\Rightarrow$ Output $\mathbf{z}^f$ is a graph signal



- Filter bank output is a collection of F graph signals $\Rightarrow$ Matrix graph signal $\mathbf{Z} = [\mathbf{z}^1, \ldots, \mathbf{z}^F]$

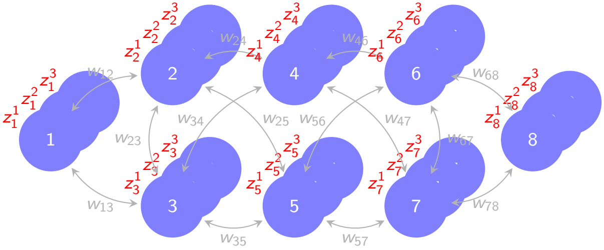
- The input of a filter bank is a single graph signal $\mathbf{x}$. Rows of $\mathbf{x}$ are signals components x_i.
- Output matrix $\mathbf{Z}$ is a collection of signals $\mathbf{z}^f$. Rows of which are components z_i^f.
- Vector $\mathbf{z}_i$ supported at each node. Columns of $\mathbf{Z}$ are graph signals $\mathbf{z}^f$. Rows of $\mathbf{Z}$ are node features $\mathbf{z}_i$



$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix}$$

Filter Bank Outputs: Multiple Features

- The input of a filter bank is a single graph signal $\mathbf{x}$. Rows of $\mathbf{x}$ are signals components x_i.
- Output matrix $\mathbf{Z}$ is a collection of signals $\mathbf{z}^f$. Rows of which are components z_i^f.
- Vector $\mathbf{z}_i$ supported at each node. Columns of $\mathbf{Z}$ are graph signals $\mathbf{z}^f$. Rows of $\mathbf{Z}$ are node features $\mathbf{z}_i$

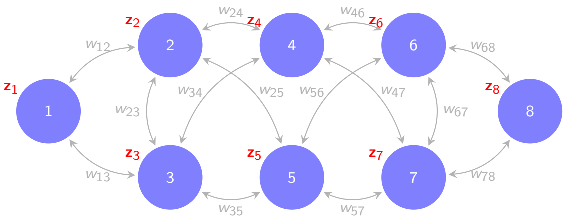


$$\mathbf{Z} = \begin{bmatrix} z_1^1 & \cdots & z_1^f & \cdots & z_1^F \\ \vdots & & \vdots & & \vdots \\ z_i^1 & \cdots & z_i^f & \cdots & z_i^F \\ \vdots & & \vdots & & \vdots \\ z_n^1 & \cdots & z_n^f & \cdots & z_n^F \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_i \\ \vdots \\ \mathbf{z}_n \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{z}^1 & \cdots & \mathbf{z}^f & \cdots & \mathbf{z}^F \end{bmatrix}$$

Filter Bank Outputs: Multiple Features

- The input of a filter bank is a single graph signal $\mathbf{x}$. Rows of $\mathbf{x}$ are signals components x_i.
- Output matrix $\mathbf{Z}$ is a collection of signals $\mathbf{z}^f$. Rows of which are components z_i^f.
- Vector $\mathbf{z}_i$ supported at each node. Columns of $\mathbf{Z}$ are graph signals $\mathbf{z}^f$. Rows of $\mathbf{Z}$ are node features $\mathbf{z}_i$



$$\mathbf{Z} = \begin{bmatrix} z_1^1 & \cdots & z_1^f & \cdots & z_1^F \\ \vdots & & \vdots & & \vdots \\ z_i^1 & \cdots & z_i^f & \cdots & z_i^F \\ \vdots & & \vdots & & \vdots \\ z_n^1 & \cdots & z_n^f & \cdots & z_n^F \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_i \\ \vdots \\ \mathbf{z}_n \end{bmatrix}$$

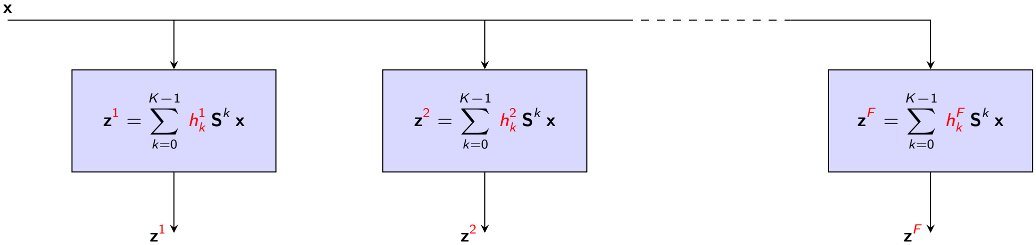
$$= \begin{bmatrix} \mathbf{z}^1 & \cdots & \mathbf{z}^f & \cdots & \mathbf{z}^F \end{bmatrix}$$

Multiple Feature GNNs

- We leverage filter banks to create GNNs that process multiple features per layer

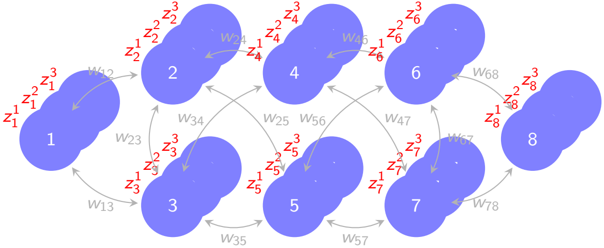
Multiple Feature (Matrix) Graph Signals

- Filter banks output a collection of multiple graph signals $\Rightarrow$ A matrix graph signal $\mathbf{Z} = [\mathbf{z}^1, \ldots, \mathbf{z}^F]$
- The F graph signals $\mathbf{z}^f$ represent F features per node. A vector $\mathbf{z}_i$ supported at each node



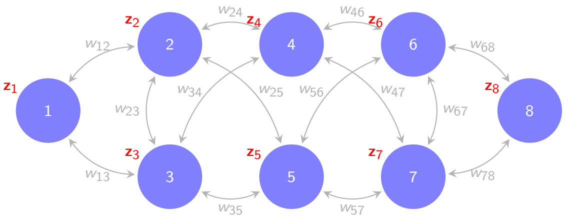
- We would now like to process multiple feature graph signals. Process each feature with a filterbank.

- Filter banks output a collection of multiple graph signals $\Rightarrow$ A matrix graph signal $\mathbf{Z} = [\mathbf{z}^1, \ldots, \mathbf{z}^F]$
- The F graph signals $\mathbf{z}^f$ represent F features per node. A vector $\mathbf{z}_i$ supported at each node



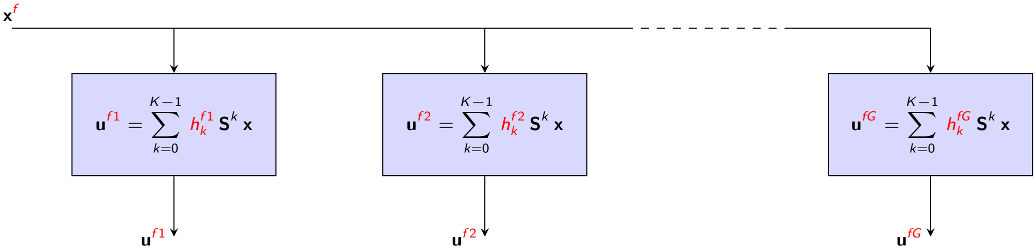
- We would now like to process multiple feature graph signals. Process each feature with a filterbank.

- Filter banks output a collection of multiple graph signals $\Rightarrow$ A matrix graph signal $\mathbf{Z} = [\mathbf{z}^1, \ldots, \mathbf{z}^F]$
- The F graph signals $\mathbf{z}^f$ represent F features per node. A vector $\mathbf{z}_i$ supported at each node

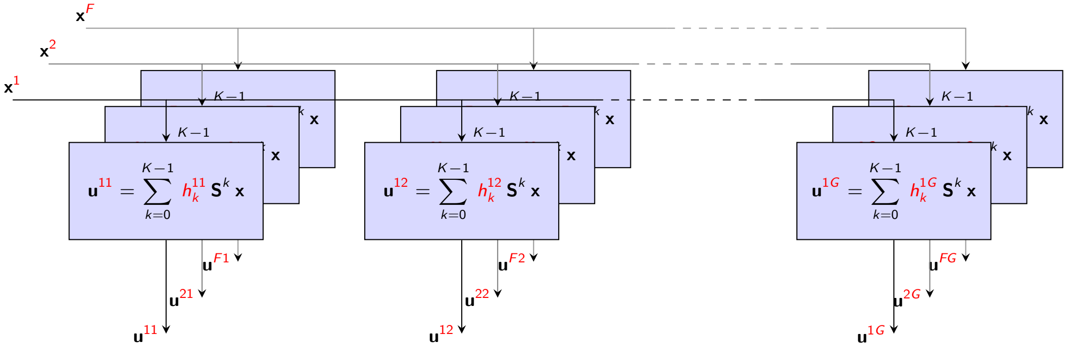


- We would now like to process multiple feature graph signals. Process each feature with a filterbank.

- Each of the F features $\mathbf{x}^f$ is processed with G filters with coefficients h_k^{fg} $\Rightarrow \mathbf{u}^{fg} = \sum_{k=0}^{K-1} h_k^{fg}\, \mathbf{S}^k\, \mathbf{x}^f$

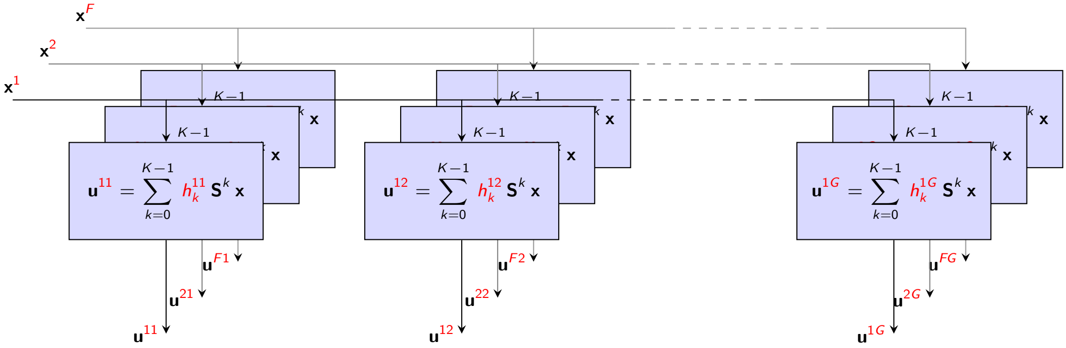


- This Multiple-Input-Multiple-Output Graph Filter generates an output with $F \times G$ features



$\mathbf{z}^1 = \mathbf{u}^{11} + \mathbf{u}^{21} + \ldots + \mathbf{u}^{F1}$ $\quad$ $\mathbf{z}^2 = \mathbf{u}^{12} + \mathbf{u}^{22} + \ldots + \mathbf{u}^{F2}$ $\quad$ $\mathbf{z}^G = \mathbf{u}^{1G} + \mathbf{u}^{2G} + \ldots + \mathbf{u}^{FG}$

- Reduce to G outputs with sum over input features for given g $\Rightarrow \mathbf{z}^g = \sum_{f=1}^{F} \mathbf{u}^{fg} = \sum_{f=1}^{F} \sum_{k=0}^{K-1} h_k^{fg} \mathbf{S}^k \mathbf{x}^f$



$\mathbf{z}^1 = \mathbf{u}^{11} + \mathbf{u}^{21} + \ldots + \mathbf{u}^{F1}$ $\quad$ $\mathbf{z}^2 = \mathbf{u}^{12} + \mathbf{u}^{22} + \ldots + \mathbf{u}^{F2}$ $\quad$ $\mathbf{z}^G = \mathbf{u}^{1G} + \mathbf{u}^{2G} + \ldots + \mathbf{u}^{FG}$

MIMO Graph Filters with Matrix Graph Signals

- MIMO graph filters are cumbersome, not difficult. Just $F \times G$ filters. Or F filter banks.
- Easier with matrices $\Rightarrow$ $G \times F$ coefficient matrix $\mathbf{H}_k$ with entries $\Big(\mathbf{H}_k\Big)_{fg} = h_k^{fg}$

$$\mathbf{Z} = \sum_{k=0}^{K-1} \mathbf{S}^k \times \mathbf{X} \times \mathbf{H}_k$$

- This is a more compact format of the MIMO filter. It is equivalent

$$\begin{bmatrix} \mathbf{z}^1 & \cdots & \mathbf{z}^g & \cdots & \mathbf{z}^G \end{bmatrix} = \sum_{k=0}^{K-1} \mathbf{S}^k \times \begin{bmatrix} \mathbf{x}^1 & \cdots & \mathbf{x}^f & \cdots & \mathbf{x}^F \end{bmatrix} \times \begin{bmatrix} h_k^{11} & \cdots & h_k^{1g} & \cdots & h_k^{1G} \\ \vdots & & \vdots & & \vdots \\ h_k^{f1} & \cdots & h_k^{fg} & \cdots & h_k^{fG} \\ \vdots & & \vdots & & \vdots \\ h_k^{F1} & \cdots & h_k^{Fg} & \cdots & h_k^{FG} \end{bmatrix}$$

- MIMO GNN stacks MIMO perceptrons $\Rightarrow$ Compose of MIMO filters with pointwise nonlinearities

- Layer ℓ processes input signal $\mathbf{X}_{\ell-1}$ with perceptron $\mathbf{H}_\ell = [\mathbf{H}_{\ell 0}, \ldots, \mathbf{H}_{\ell, K-1}]$ to produce output $\mathbf{X}_\ell$

$$\mathbf{X}_\ell = \sigma\Big[\mathbf{Z}_\ell\Big] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k\, \mathbf{X}_{\ell-1}\, \mathbf{H}_{\ell k}\right]$$

- Denoting the Layer 1 input as $\mathbf{X}_0 = \mathbf{X}$, this provides a recursive definition of a MIMO GNN

- If it has L layers, the GNN output $\Rightarrow \mathbf{X}_L \;=\; \Phi\Big(\,\mathbf{x};\, \mathbf{S}, \mathbf{H}_1, \ldots, \mathbf{H}_L\Big) \;=\; \Phi\Big(\,\mathbf{x};\, \mathbf{S}, \mathcal{H}\,\Big)$

- The filter tensor $\mathcal{H} = [\mathbf{H}_1, \ldots, \mathbf{H}_L]$ is the trainable parameter. The graph shift is prior information

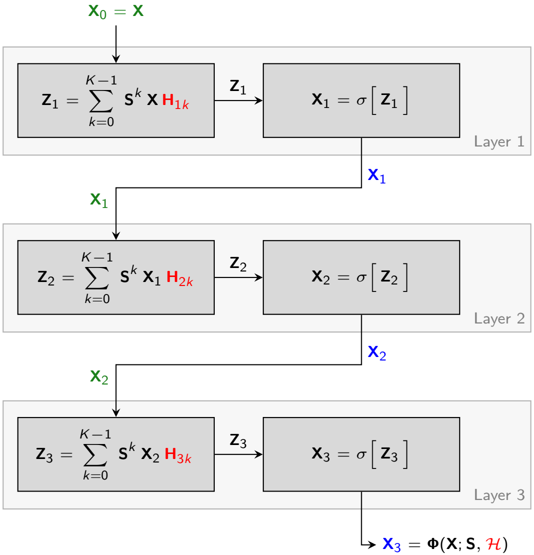
- We illustrate with a MIMO GNN with 3 layers

- Feed input signal $\mathbf{X} = \mathbf{X}_0$ into Layer 1 (F_0 features)

$$\mathbf{X}_1 = \sigma\Big[\,\mathbf{Z}_1\,\Big] = \sigma\left[\,\sum_{k=0}^{K-1} \mathbf{S}^k\, \mathbf{X}_0\, \mathbf{H}_{1k}\,\right]$$

- Last layer output is the GNN output $\Rightarrow \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H})$

  $\Rightarrow$ Parametrized by trainable tensor $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$



MIMO GNN Block Diagram

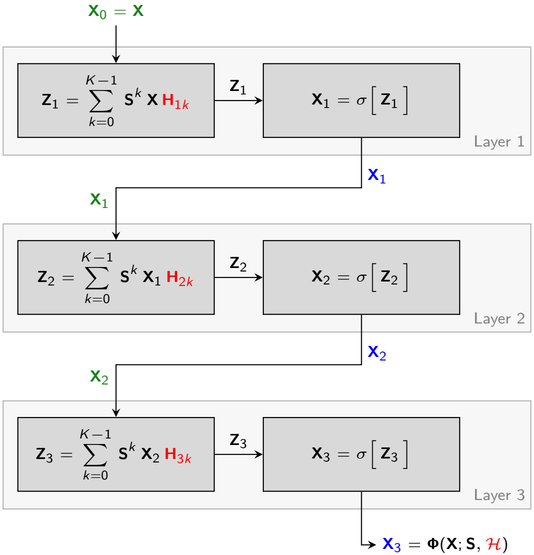
- We illustrate with a MIMO GNN with 3 layers

- Feed Layer 1 output as an input to Layer 2 (F_1 features)

$$\mathbf{X}_2 = \sigma\Big[\mathbf{Z}_2\Big] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k\,\mathbf{X}_1\,\mathbf{H}_{2k}\right]$$

- Last layer output is the GNN output $\Rightarrow \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H})$

$\Rightarrow$ Parametrized by trainable tensor $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$



MIMO GNN Block Diagram

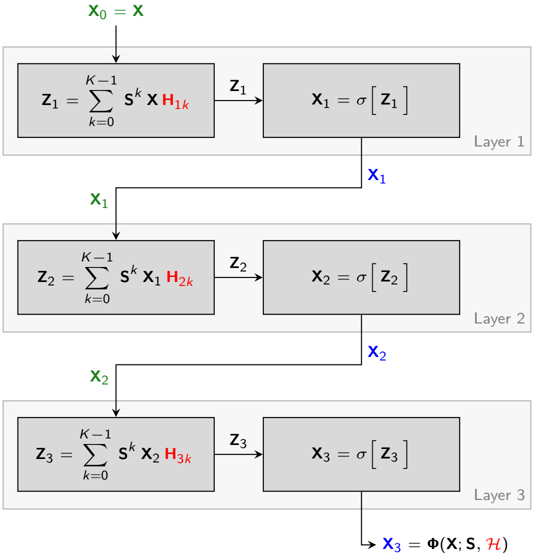
- We illustrate with a MIMO GNN with 3 layers

- Feed Layer 2 output (F_2 features) as an input to Layer 3

$$\mathbf{X}_3 = \sigma\Big[\mathbf{Z}_3\Big] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k\, \mathbf{X}_2\, \mathbf{H}_{3k}\right]$$

- Last layer output is the GNN output $\Rightarrow \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H})$

  $\Rightarrow$ Parametrized by trainable tensor $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$

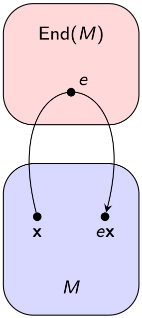


Algebraic Convolutional Information Processing

- Algebraic filters are a generic abstraction of the common features of convolutional signal processing
- Graph, time, and image convolutions can be expressed as particular cases of algebraic filters

- Signals in $M = \mathbb{R}^n \Rightarrow$ Traditional matrix multiplications $\Rightarrow \mathbf{y} = \mathbf{E}\mathbf{x}$
- Signals in $M = L_2([0,1]) \Rightarrow$ Linear functionals $\Rightarrow \mathbf{y} = \int_0^1 E(u,v)\mathbf{x}(v)\,dv$
- $\text{End}(M)$: the set of all linear maps that can be applied to a signal x in M

$\Rightarrow$ Learning in $\text{End}(M)$ is not scalable $\Rightarrow$ Search over All Matrices. Or over all linear functionals



- For Scalable learning $\Rightarrow$ We do restrict allowable linear maps

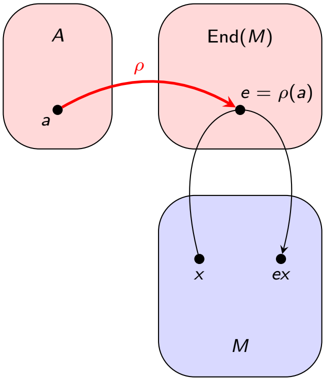
  $\Rightarrow$ To those that represent a more restrictive algebra

- Map elements a of the algebra A with a homomorphism

$$\rho : A \rightarrow \mathrm{End}(M)$$

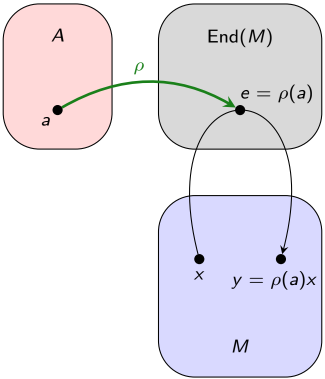
- Map abstract filters $a \in A$ into concrete endomorphisms $\rho(a)$

  $\Rightarrow$ Convolutional filters yield outputs $\Rightarrow$ $y = \rho(a)x$



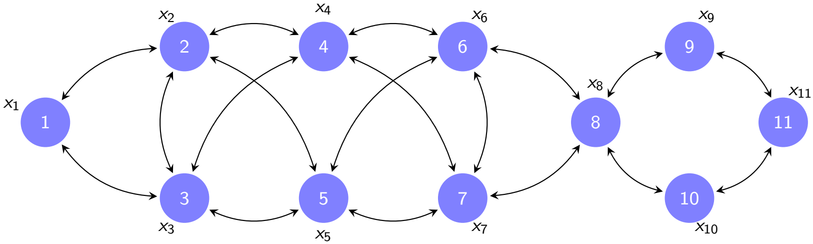
An Algebraic SP model is a triplet (A, M, ρ)

- A is an Algebra with unity where filters $a \in A$ live
- It defines the rules of convolutional signal processing
- M is a vector space
- The space containing the signals $\mathbf{x}$ we want to process
- ρ is a homomorphism from A to the endomorphisms of M



Task

Process signals $\mathbf{x}$ that are supported on a graph with n nodes. A matrix representation of the graph is given in the matrix $\mathbf{S}$.



Graph Signal Processing (GSP)

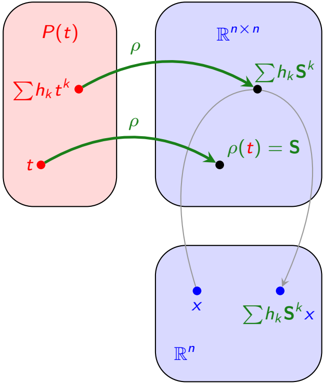
- GSP in the graph **S** is a particular case of ASP in which

  $\Rightarrow M = \mathbb{R}^n \;\Rightarrow$ Vectors with n components

  $\Rightarrow A = P(t) \;\Rightarrow$ The algebra of polynomials $h = \sum_k h_k t^k$

  $\Rightarrow$ Shift operator $\rho(t) = \mathbf{S} \;\Rightarrow$ Resulting in filters

$$\rho(h) = \rho\left(\sum_k h_k t^k\right) = \sum_k h_k \mathbf{S}^k$$



- Processing $\mathbf{x}$ with filter $\rho(h)$ yields output $\Rightarrow \mathbf{y} = \rho(h)\mathbf{x} = \rho\left(\sum_k h_k t^k\right)\mathbf{x} = \sum_k h_k \mathbf{S}^k \mathbf{x}$

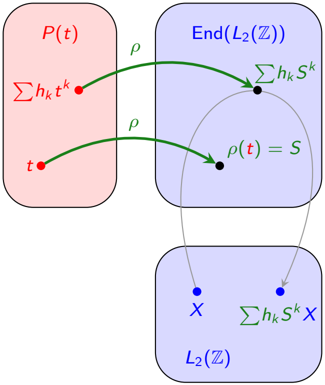
Task

Process sequences X with values $(X)_n = x_n$ for integer indexes $n \in \mathbb{Z}$. The sequences have finite energy. We say that $X \in L_2(\mathbb{Z})$

- DTSP is a particular case of ASP in which

$\Rightarrow M = L_2(\mathbb{Z}) \Rightarrow$ Finite-energy sequences in $\mathbb{Z}$

$\Rightarrow A = P(t) \Rightarrow$ The algebra of polynomials $h = \sum_k h_k t^k$



- DTSP is a particular case of ASP in which

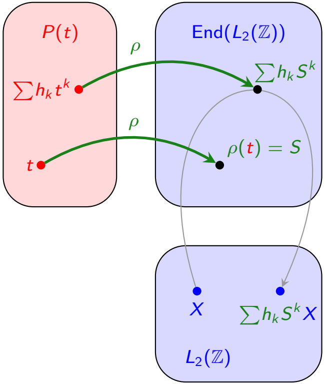
$\Rightarrow$ Shift operator is a time shift $\rho(t) = S$ such that

$$(SX)_n = (X)_{n-1}$$

$\Rightarrow$ This mapping of the generator t yields filters

$$\rho(h) \;=\; \rho\left(\sum_k h_k t^k\right) \;=\; \sum_k h_k S^k$$

where S^k represents k applications of S



- Processing X with $\rho(h)$ yields $\Rightarrow \Big(Y\Big)_n \;=\; \Big(\rho(h)X\Big)_n \;=\; \left(\sum_k h_k S^k X\right)_n \;=\; \sum_k h_k \Big(X\Big)_{n-k}$

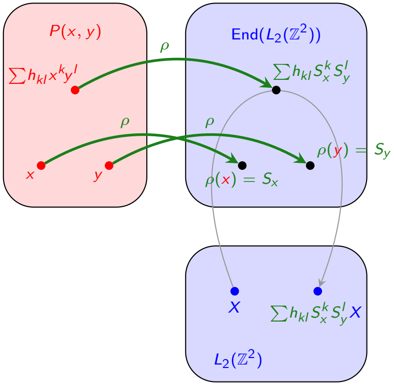
Task

Process images, defined as sequences X with values $(X)_{mn} = x_{mn}$ that depend on two integer indexes $m, n \in \mathbb{Z}$. The sequences have finite energy. We say that $X \in L_2(\mathbb{Z}^2)$

- IP is a particular case of ASP in which

  $\Rightarrow M = L_2(\mathbb{Z}^2) \Rightarrow$ Finite-energy sequences in $\mathbb{Z}^2$

  $\Rightarrow A = P(x, y) \Rightarrow$ Two-letter polynomials $h = \sum_k h_{kl} x^k y^l$



- IP is a particular case of ASP in which

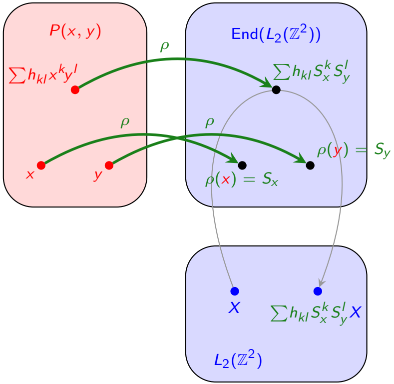
$\Rightarrow$ Two shift operators $\rho(x) = S_x$ and $\rho(y) = S_y$

$$(S_x X)_{mn} = (X)_{(m-1)n} \quad (S_y X)_{mn} = (X)_{m(n-1)}$$

$\Rightarrow$ This mapping of the generators x and y yields filters

$$\rho(h) \;=\; \rho\left(\sum_k h_k t^k\right) \;=\; \sum_k h_{kl} S_x^k S_y^l$$

S_x^k or S_x^k represent k or l applications of S_x orS_l

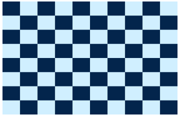


- Processing X yields $\Rightarrow \big(Y\big)_{mn} \;=\; \big(\rho(h)X\big)_{mn} \;=\; \left(\sum_{kl} h_{kl} S_x^k S_y^l X\right)_{mn} \;=\; \sum_k h_k \big(X\big)_{(m-k)(n-l)}$

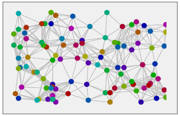
ASP is a Generic Analysis Tool

- Algebraic SP encompasses Graph SP, graphon SP, Time SP, and Image SP as particular cases

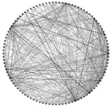
$\Rightarrow$ Other particular cases exist. Notably, Group SP



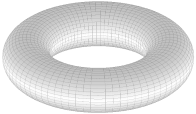
Euclidean



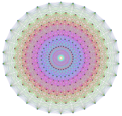
Graph



Graphons



Manifolds



Lie Groups

- ASP provides a framework for fundamental analyses that hold for all forms of convolutional filters

Generators, Shift Operators, and Frequency Representations

- Algebraic Signal Processing is an abstraction of Convolutional Information Processing

- Three central components $\Rightarrow$ generators, shift operators, and frequency representations

Definition (Generators)

The set $\mathcal{G} \subseteq A$ generates A if all $h \in A$ can be represented as polynomials of the elements of $\mathcal{G}$,

$$h \;=\; \sum_{k_1,\ldots k_r} h_{k_1,\ldots,k_r}\, {g_1}^{k_1} \ldots {g_r}^{k_r} \;=\; p_A(\mathcal{G})$$

- The elements $g \in \mathcal{G}$ are the generators of A. And $h = p_A(\mathcal{G})$ is the polynomial that generates h

  $\Rightarrow$ Filters can be built from the generating set using the operations of the algebra

- Given the algebra, the generators are given $\Rightarrow$ Filter h is completely specified by its coefficients

- The algebra of polynomials of a single variable t is generated by the polynomial $g = t$

$\Rightarrow$ Algebra elements are expressions $h = \sum_k h_k t^k$ $\Rightarrow$ They can be generated as $h = \sum_k h_k t^k$

- Algebra of polynomials of two variables x and y is generated by the polynomials $g_1 = x$ and $g_2 = y$

$\Rightarrow$ Algebra elements are expressions $h = \sum_k h_{kl} x^k y^l$ $\Rightarrow$ Can be generated as $h = \sum_k h_{kl} x^k y^l$

Definition (Shift Operators)

Let (M, ρ) be a representation of A and $\mathcal{G} \subseteq A$ a generator set of A. We say $\mathbf{S}$ is a shift operator if

$$\mathbf{S} = \rho(g), \quad \text{for some } g \in \mathcal{G}$$

The set $\mathcal{S} = \{\rho(g),\ g \in \mathcal{G}\}$ is the shift operator set of the representation (M, ρ) of algebra A.

- Generators g of Algebra A mapped to shift operators $\mathbf{S}$ in the space $\text{End}(M)$ of endomorphisms of M

Theorem (Filters as Polynomials on Shift Operators)

Let (M, ρ) be a representation of A with generators $g_i \in \mathcal{G}$ and shift operators $\mathbf{S}_i = \rho(g_i) \in \mathcal{S}$.

The representation $\rho(h)$ of filter h is a polynomial on the shift operator set,

$$h \;=\; p_A(\mathcal{G}) \;=\; \sum_{k_1,\ldots k_r} h_{k_1,\ldots,k_r}\, {g_1}^{k_1} \ldots {g_r}^{k_r} \quad\Rightarrow\quad \rho(h) \;=\; p_M(\mathcal{S}) \;=\; \sum_{k_1,\ldots k_r} h_{k_1,\ldots,k_r}\, {\mathbf{S}_1}^{k_1} \ldots {\mathbf{S}_r}^{k_r}$$

- GSP $\Rightarrow$ Signals in $\mathbb{R}^n$ + Algebra of Polynomials + Homomorphism ρ defined by the map

$$h = \sum_{k=0}^{K} h_k t^k \quad \rightarrow \quad \rho(h) = \sum_{k=0}^{K} h_k \mathbf{S}^k$$

- Equivalent to the (much) simpler specification of the homomorphism $\Rightarrow \rho(t) = \mathbf{S}$

$\Rightarrow$ This is possible because the algebra of polynomials is generated by $g = t$

Definition (Frequency Representation)

In an algebra A with generators $g_i \in \mathcal{G}$ we are given the filter h expressed as the polynomial

$$h = \sum_{k_1,\ldots k_r} h_{k_1,\ldots,k_r} \, g_1{}^{k_1} \ldots g_r{}^{k_r} = p_A(\mathcal{G})$$

The frequency representation of h over the field F[1] is the polynomial function with variables $\lambda_i \in \mathcal{L}$

$$p_F(\mathcal{L}) = \sum_{k_1,\ldots k_r} h_{k_1,\ldots,k_r} \, \lambda_1{}^{k_1} \ldots \lambda_r{}^{k_r}$$

- The two polynomials are different creatures $\Rightarrow$ The frequency representation is a simpler object

[1] The field is unspecified in the definition. But unless otherwise noted F refers to the field over which the vector space M is defined

Three Polynomials (or More)

- The central components of an ASP model are three different polynomials

  $\Rightarrow$ The filter. The filter's instantiation on the space of endomorphisms The frequency response

- These three polynomials have the same coefficients. They are related. But similar though they look

  $\Rightarrow$ They are different objects. They utilize different operations. They have different meanings.

P1: The Filter

- A polynomial on the elements g_i of the generator set $\mathcal{G}$ of the algebra A

$$p_A(\mathcal{G}) \;=\; \sum_{k_1,\ldots k_r} h_{k_1,\ldots,k_r}\, {g_1}^{k_1} \ldots {g_r}^{k_r}$$

- Sum, product, and scalar product are the operations of the algebra A
- The abstract definition of a filter. Untethered to a specific signal model

P2: The Instantiation of the Filter in the space of Endomorphisms End(M)

- A polynomial on the elements $\mathbf{S}_i = \rho(g_i)$ of the shift operator set $\mathcal{S}$

$$p_M(\mathcal{S}) \;=\; \sum_{k_1,\ldots k_r} h_{k_1,\ldots,k_r}\, \mathbf{S}_1{}^{k_1} \ldots \mathbf{S}_r{}^{k_r}$$

- Sum, product, and scalar product are the operations of the algebra of Endomorphisms of M
- The concrete effect that a filter has on a signal $\mathbf{x}$. Tethered to a specific signal model

- "Or more" $\Rightarrow$ The same abstract filter can be instantiated in multiple signal models

P3: The Frequency Response

- A polynomial function where the variables $\lambda_i \in \mathcal{L}$ take values on the field F

$$p_F(\mathcal{L}) = \sum_{k_1,\ldots k_r} h_{k_1,\ldots,k_r} \lambda_1{}^{k_1} \ldots \lambda_r{}^{k_r}$$

- Sum and product are the operations of field F. $\Rightarrow$ E.g, a polynomial with real variables
- Simpler representation of a filter. Untethered to a specific signal model (except for the field)

- The tool we use for analysis. $\Rightarrow$ To explain discriminability, stability and transferability

The Three (or More) Polynomials in GSP

(P1) Abstract filter $\Rightarrow p_A(t) = \sum_{k=0}^{K} h_k t^k$. Abstract definition. Untethered to any specific graph

(P2) Filter instantiated on a graph $\Rightarrow p_M(\mathbf{S}) = \sum_{k=0}^{K} h_k \mathbf{S}^k$. Concrete instantiation. Tethered to $\mathbf{S}$

$\Rightarrow$ On another graph $\Rightarrow p_M(\hat{\mathbf{S}}) = \sum_{k=0}^{K} h_k \hat{\mathbf{S}}^k$. Concrete instantiation. Tethered to $\hat{\mathbf{S}}$

$\Rightarrow$ On a graphon $\Rightarrow p_M(T_W) = \sum_{k=0}^{K} h_k T_W^{(k)}$. Concrete instantiation. Tethered to graphon W

(P3) Frequency response $\Rightarrow p_F(\lambda) = \sum_{k=0}^{K} h_k \lambda^k$. Simple function of one variable. Same for all instances

Algebraic Neural Networks

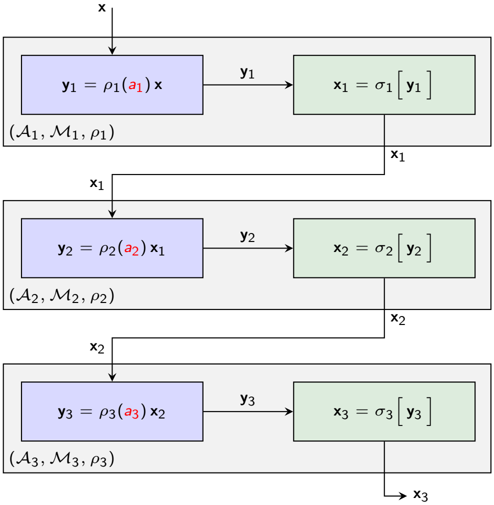
- We introduce Algebraic Neural Networks (AlgNNs) to generalize neural convolutional networks

[1] A. Parada-Mayorga and A. Ribeiro, "Algebraic Neural Networks: Stability to Deformations," IEEE TSP. ArXiv: 2009.01433.
[2] Parada-Mayorga, et al . Convolutional filtering and neural networks with non commutative algebras. ArXiv: 2108.09923.

- AlgNN is a stacked layered structure.
- Each layer: algebraic signal model $(\mathcal{A}_\ell, \mathcal{M}_\ell, \rho_\ell)$
- Mapping from layer ℓ to $\ell + 1$

$$\mathbf{x}_\ell = \sigma\left[\mathbf{y}_\ell\right] = \sigma\left[\rho_\ell\left(a_\ell\right)\mathbf{x}_{\ell-1}\right]$$

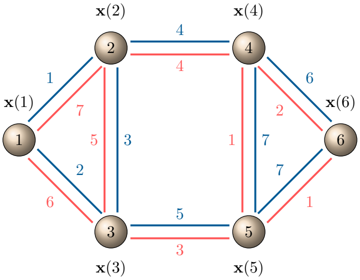
- σ_ℓ is a pointwise nonlinearity.



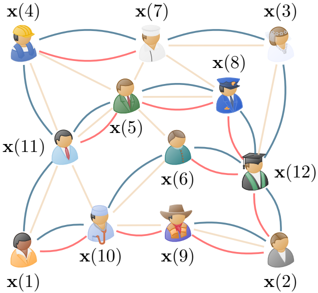
Multigraph Neural Networks

Butler, L., Parada-Mayorga, A., and Ribeiro, A. "Convolutional learning on multigraphs.", arXiv:2209.11354 (2022) TSP - IEEE.

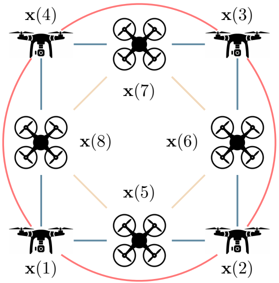
- Some networked systems emerge naturally modeled/defined by multiple types of connections



Multigraphs



Social Networks



Autonomous systems

- Multigraph $(\mathcal{V}, \{\mathcal{E}_1, \mathcal{E}_2\})$ is composed by the graphs $(\mathcal{V}, \mathcal{E}_1)$ and $(\mathcal{V}, \mathcal{E}_2)$ with the same node set $\mathcal{V}$
- Built signal models $\Rightarrow$ convolutional filtering + convolutional NN $\Rightarrow$ preserving structure

Multigraph $(\mathcal{V}, \{\mathcal{E}_1, \mathcal{E}_2\})$ is composed by the graphs $(\mathcal{V}, \mathcal{E}_1) \rightarrow \mathbf{S}_1$ and $(\mathcal{V}, \mathcal{E}_2) \rightarrow \mathbf{S}_2$ with the same node set $\mathcal{V}$

- Signals, $\mathbf{x}$, are elements of $\mathbb{R}^N$, $N = |\mathcal{V}|$. The i-th component of $\mathbf{x}$ lives on node $i \in \mathcal{V}$
- The algebra $\mathcal{A}$ $\Rightarrow$ The algebra of polynomials with independent variables t_1, t_2 (non commutative)
- The homomorphism ρ translates the polynomials $h(t_1, t_2)$ into the matrix polynomials $\mathbf{H}(\mathbf{S}_1, \mathbf{S}_2)$

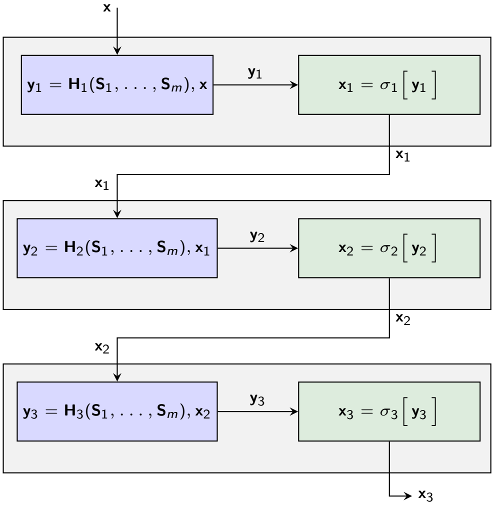
$$\mathbf{H}(\mathbf{S}_1, \mathbf{S}_2) = \mathbf{S}_1^2 + \mathbf{S}_2\mathbf{S}_1 + 2\mathbf{S}_2^2 + \mathbf{I}$$

- Multigraph NN is a stacked layered structure

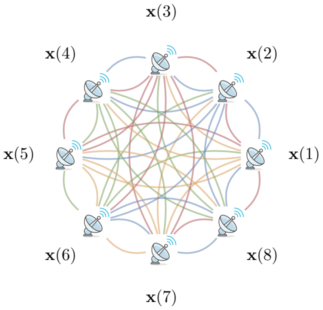
- Mapping from layer ℓ to $\ell+1$

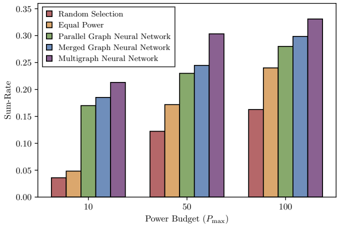
$$\mathbf{x}_\ell = \sigma\Big[\mathbf{y}_\ell\Big] = \sigma\Big[\mathbf{H}_\ell(\mathbf{S}_1, \ldots, \mathbf{S}_m)\mathbf{x}_{\ell-1}\Big]$$

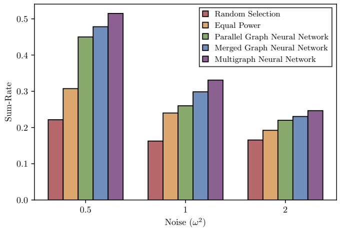
- Learnable parameters: coefficients of $\mathbf{H}$



- Multigraph Neural Networks capture complex network dynamics that graphs and GNNs cannot







Power allocation problem in a wireless communication system

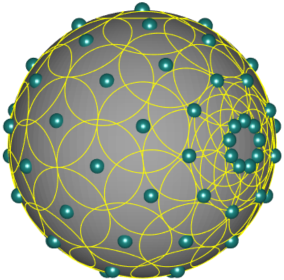
- Diffusions capture/exploit heterogeneous structure

Butler, L., Parada-Mayorga, A., and Ribeiro, A. "Convolutional learning on multigraphs.", arXiv:2209.11354 (2022) TSP - IEEE.

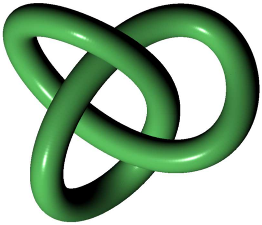
Lie Algebra Group Neural Networks

Kumar, H., Parada-Mayorga, A., & Ribeiro, A. (2023). Lie Group Algebra Convolutional Filters. arXiv: 2305.04431

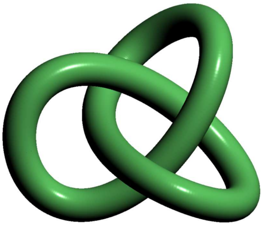
- Lie group symmetries on arbitrary spaces: no lifting on the group, non homogeneous spaces
- Proteins and knot type data $\Rightarrow$ locally complex, low dimensional in high dimensional spaces



Rotation symmetries $SO(3)$



Group action 1



Group action 2

- non homogeneous spaces: concrete real life data defined from arbitrary/irregular sampling schemes
- Built signal models $\Rightarrow$ convolutional filtering + convolutional NN $\Rightarrow$ preserving structure

- Signals, $\mathbf{x} \in \mathcal{M} \Rightarrow$ arbitrary (given) vector space. No need to lift information onto the group
- The algebra $\mathcal{A} = L^1(\widehat{G})$ is a polynomial algebra constructed from generators of $L^1(G)$)
- The homomorphism instantiates convolutional filters as multivariate polynomials (for example with 2 generators)

$$\mathbf{H}\left(\widehat{\mathbf{T}}_{g_1}, \widehat{\mathbf{T}}_{g_2}\right) = \widehat{\mathbf{T}}_{g_1}^2 + \widehat{\mathbf{T}}_{g_1}\widehat{\mathbf{T}}_{g_2} + 2\widehat{\mathbf{T}}_{g_2}^2 + \mathbf{I}$$

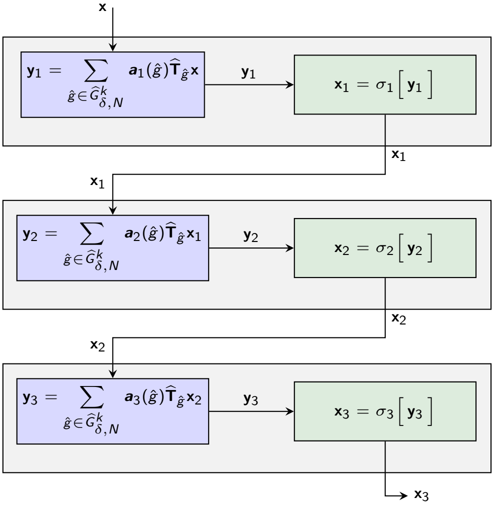
- $\widehat{\mathbf{T}}_{g_1}, \widehat{\mathbf{T}}_{g_2}$ actions of the generators of $L^1(G)$ on the space of signals $\mathcal{M}$

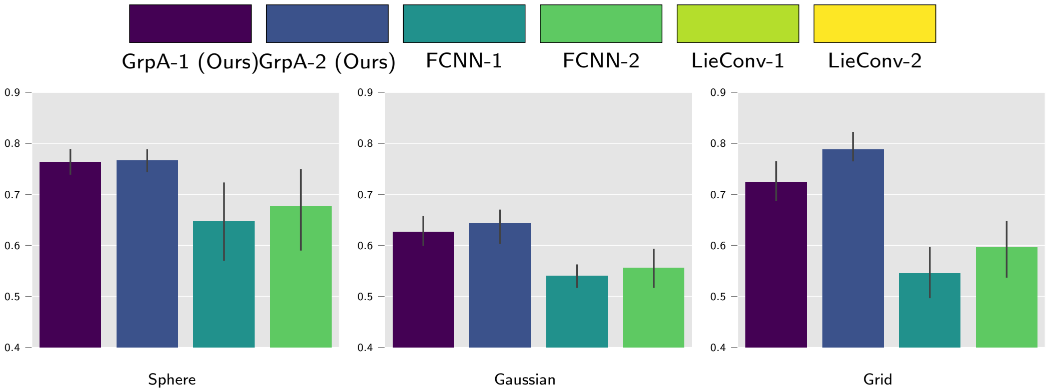
- Lie group Algebra NN: stacked layered structure

- Mapping from layer ℓ to $\ell+1$

$$\mathbf{x}_\ell = \sigma\left[\mathbf{y}_\ell\right] = \sigma_\ell\left[\mathbf{H}_\ell\left(\widehat{\mathbf{T}}_{g_1},\ldots,\widehat{\mathbf{T}}_{g_m}\right)\mathbf{x}_{\ell-1}\right]$$

- Learnable parameters: coefficients of $\mathbf{H}$





Test accuracy on binary knot classification. Signals defined on Sphere, Gaussian and Uniform grids with many samples ($|\widehat{\mathcal{X}}| = 1000$). Simulations for LieConv-1,2 (intractability).

Kumar, H., Parada-Mayorga, A., & Ribeiro, A. (2023). Lie Group Algebra Convolutional Filters. arXiv: 2305.04431