Graphs

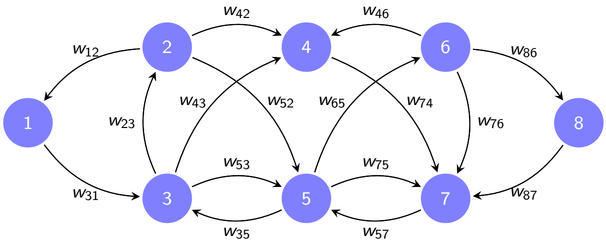
Nodes, Edges and Weights

- A graph is a triplet $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, which includes vertices $\mathcal{V}$, edges $\mathcal{E}$, and weights $\mathcal{W}$

  ⇒ Vertices or nodes are a set of n labels. Typical labels are $\mathcal{V} = \{1, \ldots, n\}$

  ⇒ Edges are ordered pairs of labels (i, j). We interpret $(i, j) \in \mathcal{E}$ as "i can be influenced by j."

  ⇒ Weights $w_{ij} \in \mathbb{R}$ are numbers associated to edges (i, j). "Strength of the influence of j on i."

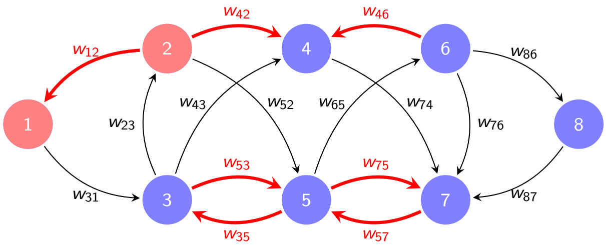


- Edge (i, j) is represented by an arrow pointing from j into i. Influence of node j on node i

  $\Rightarrow$ This is the opposite of the standard notation used in graph theory

- Edge (i, j) is different from edge (j, i) $\Rightarrow$ It is possible to have $(i, j) \in .\mathcal{E}$ and $(j, i) \notin \mathcal{E}$

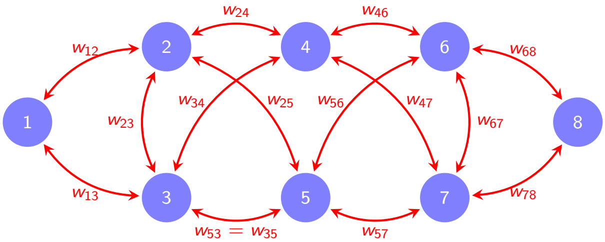
- If both edges are in the edge set, the weights can be different $\Rightarrow$ It is possible to have $w_{ij} \neq w_{ji}$



- 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 $(j, i) \in \mathcal{E}$

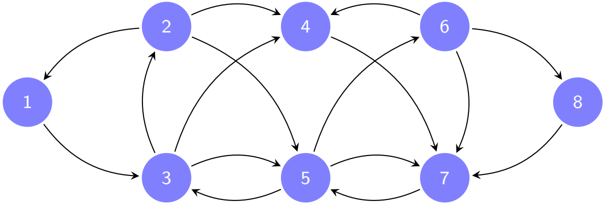
$\Rightarrow$ Weights are symmetric $\Rightarrow$ We must have $w_{ij} = w_{ji}$ for all $(i, j) \in \mathcal{E}$



- A graph is unweighted if it doesn't have weights

  $\Rightarrow$ Equivalently, we can say that all weights are units $\Rightarrow w_{ij} = 1$ for all $(i, j) \in \mathcal{E}$

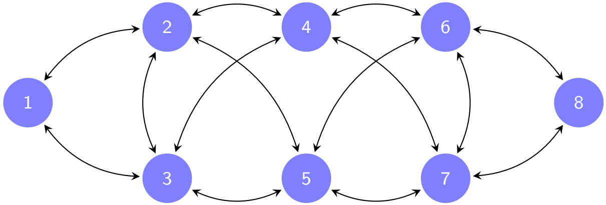
- Unweighted graphs could be directed or symmetric



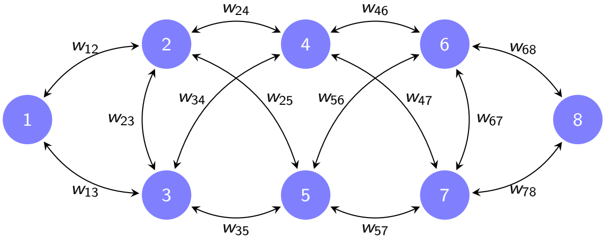
- A graph is unweighted if it doesn't have weights

$\Rightarrow$ Equivalently, we can say that all weights are units $\Rightarrow w_{ij} = 1$ for all $(i, j) \in \mathcal{E}$

- Unweighted graphs could be directed or symmetric



- Graphs can be directed or symmetric. Separately, they can be weighted or unweighted.
- Most of the graphs we encounter in practical situations are symmetric and weighted



Graph Shift Operators

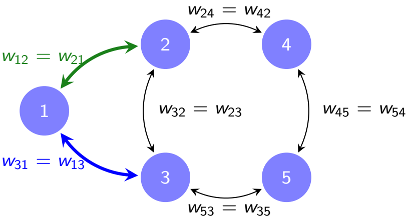
- Graphs have matrix representations. Which in this course, we call graph shift operators (GSOs)

Adjacency Matrices

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

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

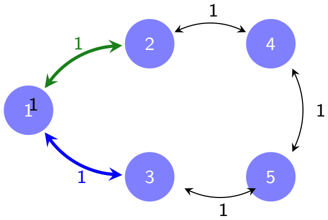
- If the graph is symmetric, the adjacency matrix is symmetric $\Rightarrow \mathbf{A} = \mathbf{A}^T$. As in the example



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

- For the particular case in which the graph is unweighted. Weights interpreted as units

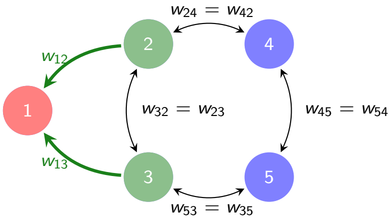
$$A_{ij} = 1, \qquad \text{for all} \quad (i, j) \in \mathcal{E}$$



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

- 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 $\Rightarrow d_i = \sum_{j \in n(i)} w_{ij} = \sum_{j:(i,j) \in \mathcal{E}\}} w_{ij}$

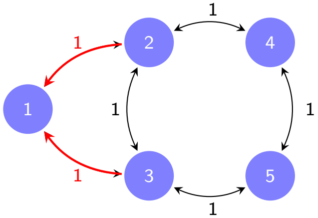


- Node 1 neighborhood $\Rightarrow n(1) = \{2, 3\}$

- 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 $(\mathbf{A1})_i = \sum_j w_{ij} = d_i$



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

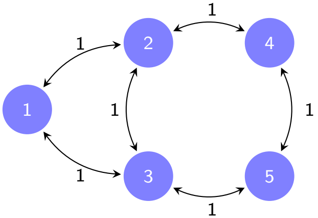
- The Laplacian matrix of a graph with adjacency matrix $\mathbf{A}$ is $\Rightarrow \mathbf{L} = \mathbf{D} - \mathbf{A} = \text{diag}(\mathbf{A1}) - \mathbf{A}$

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

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



Normalized Matrix Representations: Adjacencies

- Normalized adjacency and Laplacian matrices express weights relative to the nodes' degrees

- Normalized adjacency matrix $\Rightarrow \bar{\mathbf{A}} := \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ $\Rightarrow$ Results in entries $(\bar{\mathbf{A}})_{ij} = \dfrac{w_{ij}}{\sqrt{d_i d_j}}$

- The normalized adjacency is symmetric if the graph is symmetric $\Rightarrow \bar{\mathbf{A}}^T = \bar{\mathbf{A}}$.

- Normalized Laplacian matrix $\Rightarrow \bar{\mathbf{L}} := \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. Same normalization of adjacency matrix

- Given definitions normalized representations $\Rightarrow \bar{\mathbf{L}} = \mathbf{D}^{-1/2}\Big(\mathbf{D} - \mathbf{A}\Big)\mathbf{D}^{-1/2} = \mathbf{I} - \bar{\mathbf{A}}$

  $\Rightarrow$ The normalized Laplacian and adjacency are essentially the same linear transformation.

- Normalized operators are more homogeneous. The entries in the vector **A1** tend to be similar.

- 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
$\mathbf{S} = \mathbf{A}$	$\mathbf{S} = \mathbf{L}$	$\mathbf{S} = \bar{\mathbf{A}}$	$\mathbf{S} = \bar{\mathbf{L}}$

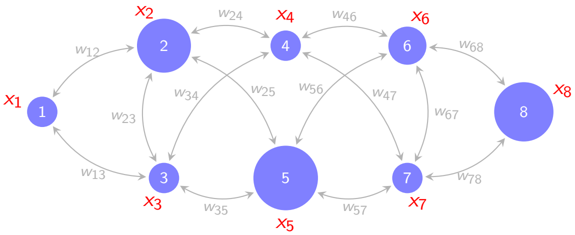
- If the graph is symmetric, the shift operator $\mathbf{S}$ is symmetric $\Rightarrow \mathbf{S} = \mathbf{S}^T$

- 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

- 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 $\Rightarrow (\mathbf{S}, \mathbf{x})$

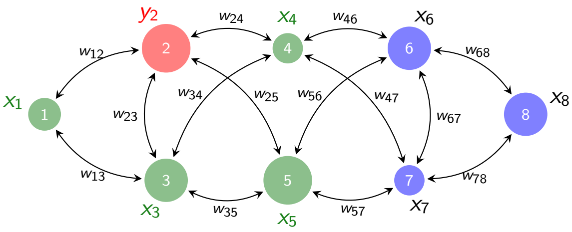


- The graph is an expectation of proximity or similarity between components of the signal $\mathbf{x}$

- 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.

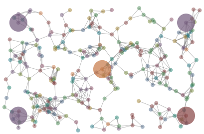


The Diffusion Sequence

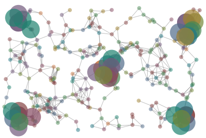
- Compose the diffusion operator to produce diffusion sequence $\Rightarrow$ defined recursively as

$$\mathbf{x}^{(k+1)} = \mathbf{S}\mathbf{x}^{(k)}, \qquad \text{with} \quad \mathbf{x}^{(0)} = \mathbf{x}$$

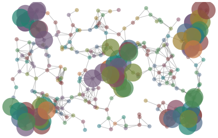
- Can unroll the recursion and write the diffusion sequence as the power sequence $\Rightarrow \mathbf{x}^{(k)} = \mathbf{S}^k\mathbf{x}$



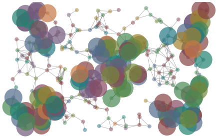
$\mathbf{x}^{(0)} = \mathbf{x} \;\; = \mathbf{S}^0\mathbf{x}$



$\mathbf{x}^{(1)} = \mathbf{S}\mathbf{x}^{(0)} \;\; = \mathbf{S}^1\mathbf{x}$



$\mathbf{x}^{(2)} = \mathbf{S}\mathbf{x}^{(1)} \;\; = \mathbf{S}^2\mathbf{x}$



$\mathbf{x}^{(3)} = \mathbf{S}\mathbf{x}^{(2)} \;\; = \mathbf{S}^3\mathbf{x}$

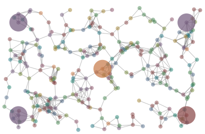
Some Observations about the Diffusion Sequence

- The kth element of the diffusion sequence $x^{(k)}$ diffuses information to k-hop neighborhoods

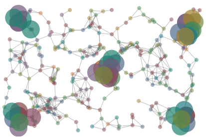
  $\Rightarrow$ One reason why we use the diffusion sequence to define graph convolutions

- We have two definitions. One recursive. The other one using powers of $\mathbf{S}$

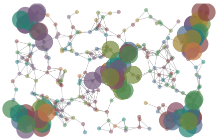
  $\Rightarrow$ Always implement the recursive version. The power version is good for analysis



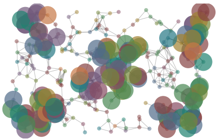
$\mathbf{x}^{(0)} = \mathbf{x} = \mathbf{S}^0\mathbf{x}$



$\mathbf{x}^{(1)} = \mathbf{S}\mathbf{x}^{(0)} = \mathbf{S}^1\mathbf{x}$



$\mathbf{x}^{(2)} = \mathbf{S}\mathbf{x}^{(1)} = \mathbf{S}^2\mathbf{x}$



$\mathbf{x}^{(3)} = \mathbf{S}\mathbf{x}^{(2)} = \mathbf{S}^3\mathbf{x}$

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 $\mathbf{S}$

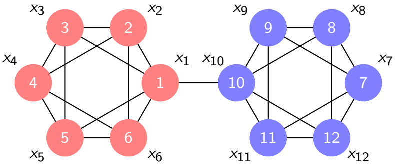
$$\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{\infty} h_k \mathbf{S}^k$$

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

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

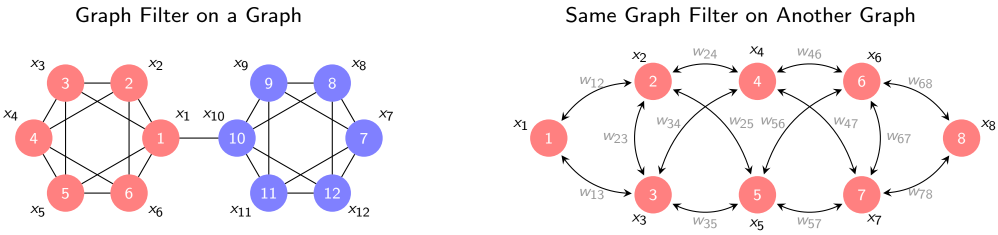
- We say that $\mathbf{y} = \mathbf{h} \star_{\mathbf{S}} \mathbf{x}$ is the graph convolution of the filter $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$ with the signal $\mathbf{x}$

- Graph convolutions aggregate information growing from local to global neighborhoods
- Consider a signal $\mathbf{x}$ supported on a graph with shift operator $\mathbf{S}$. Along with filter $\mathbf{h} = \{h_k\}_{k=0}^{K-1}$



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

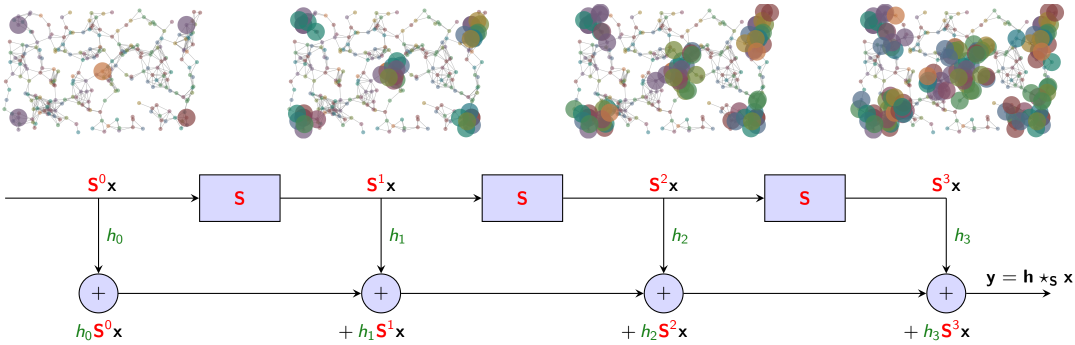
- The same filter $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$ can be executed in multiple graphs $\Rightarrow$ We can transfer the filter



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

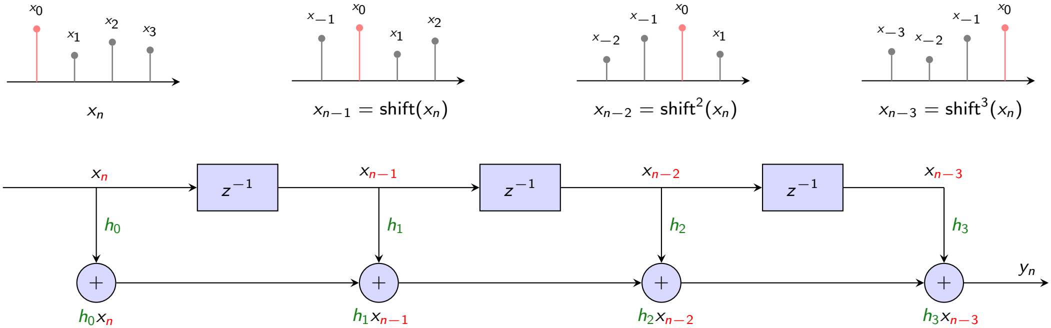
- Output depends on the filter coefficients $\mathbf{h}$, the graph shift operator $\mathbf{S}$ and the signal $\mathbf{x}$

- 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



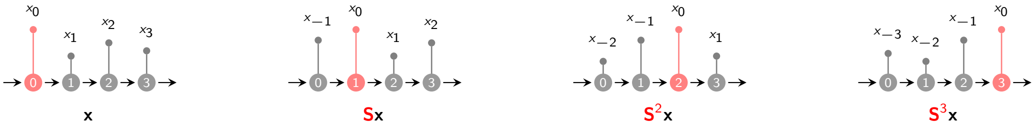
Time Convolutions as a Particular Case of Graph Convolutions

- 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 $(\mathbf{S}, \mathbf{x})$



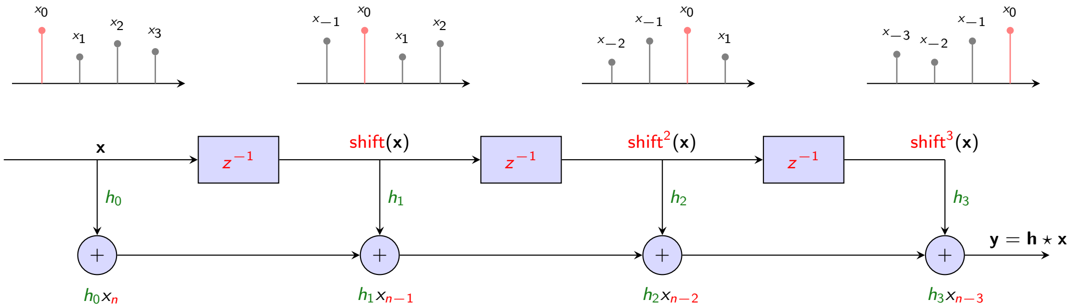
- Time shift is reinterpreted as multiplication by the adjacency matrix $\mathbf{S}$ of the line graph

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

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

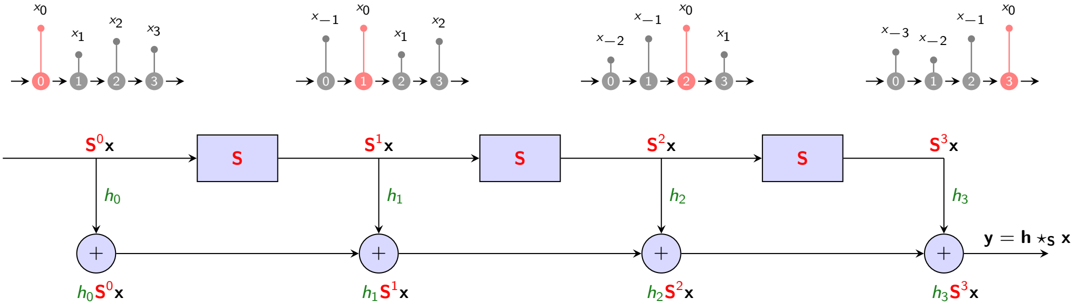
$\Rightarrow$ We can rewrite convolutional filters as polynomials on $\mathbf{S}$, the adjacency of the line graph

- 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



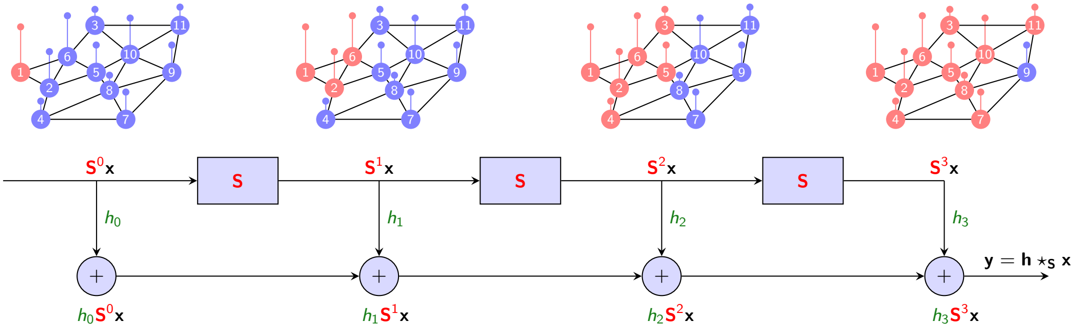
- Time convolution is a polynomial on adjacency matrix of line graph $\Rightarrow \mathbf{y} = \mathbf{h} \star \mathbf{x} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$

- 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 **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 $\mathbf{\Lambda}$

Proof: Since $\mathbf{S} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^H$, can write shift operator powers as $\mathbf{S}^k = \mathbf{V}\boldsymbol{\Lambda}^k\mathbf{V}^H$. Therefore filter output is

$$\mathbf{y} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x} = \sum_{k=0}^{\infty} h_k \mathbf{V}\boldsymbol{\Lambda}^k\mathbf{V}^H \mathbf{x}$$

- Multiply both sides by $\mathbf{V}^H$ on the left $\Rightarrow \mathbf{V}^H\mathbf{y} = \mathbf{V}^H \sum_{k=0}^{\infty} h_k \mathbf{V}\boldsymbol{\Lambda}^k\mathbf{V}^H \mathbf{x}$

- Copy and identify terms. Output GFT $\mathbf{V}^H\mathbf{y} = \tilde{\mathbf{y}}$. Input GFT $\mathbf{V}^H\mathbf{x} = \tilde{\mathbf{x}}$. Cancel out $\mathbf{V}^H\mathbf{V}$

$$\mathbf{V}^H\mathbf{y} = \mathbf{V}^H \sum_{k=0}^{\infty} h_k \mathbf{V}\boldsymbol{\Lambda}^k\mathbf{V}^H \mathbf{x} \quad \Rightarrow \quad \tilde{\mathbf{y}} = \sum_{k=0}^{\infty} h_k \boldsymbol{\Lambda}^k \tilde{\mathbf{x}}$$

■

- 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

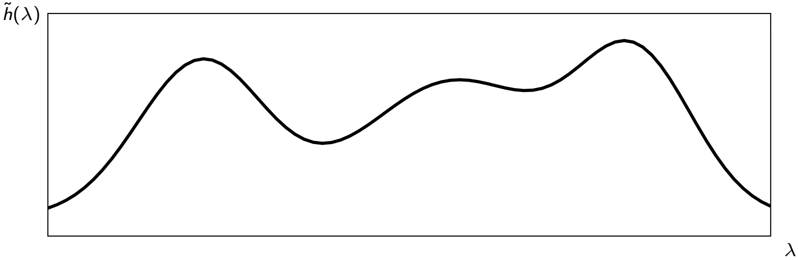
$$\tilde{h}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$$

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

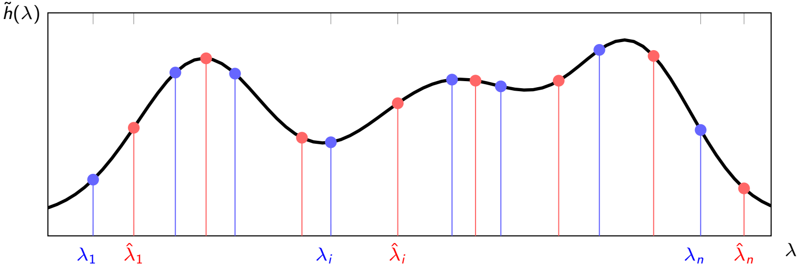
- Completely determined by the filter coefficients $\mathbf{h} = \{h_k\}_{k=1}^{\infty}$. The Graph has nothing to do with it



The Graph Instantiates Specific Eigenvalues

- 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

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

- 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

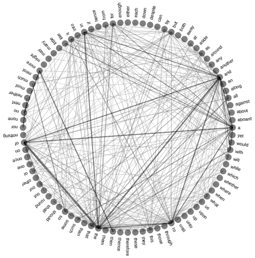
Empirical Risk Minimization with Graph Signals

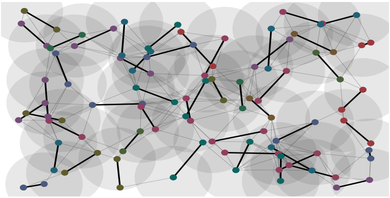
- In ERM, the function class $\mathcal{C}$ is the degree of freedom available to the system's designer

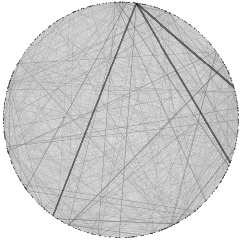
$$\Phi^* = \underset{\Phi \in \mathcal{C}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{y}, \Phi(\mathbf{x})\Big)$$

- 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



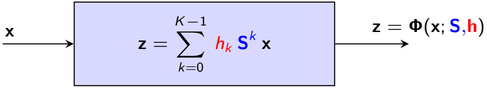




Learning with a Graph Convolutional Filter

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



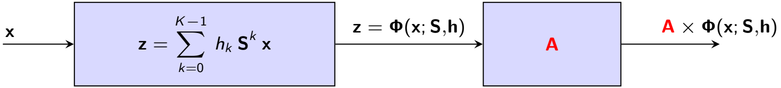
- Learn ERM solution restricted to graph filter class $\Rightarrow$ $\mathbf{h}^* = \underset{\mathbf{h}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{y}, \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h})\Big)$

  $\Rightarrow$ 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. $\Rightarrow \mathbf{h}^* = \underset{\mathbf{h}}{\text{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{y}, \mathbf{A} \times \Phi(\mathbf{x};\mathbf{S},\mathbf{h})\Big)$

- 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)

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

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

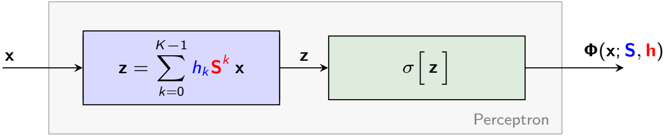
- A pointwise nonlinearity is the simplest nonlinear function we can apply to a vector

- 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.

- 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 $\mathbf{x}_L$

- 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

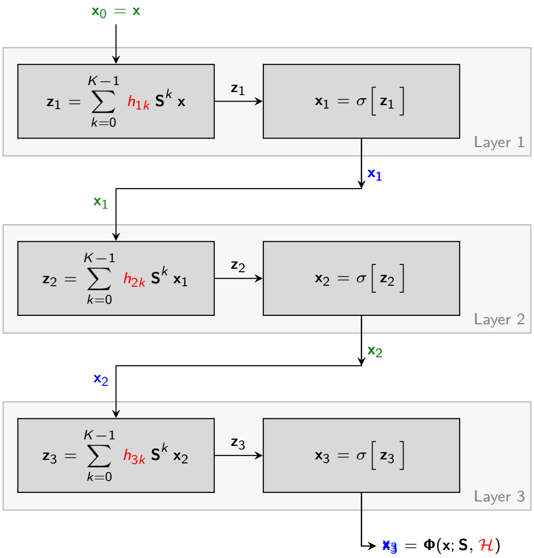
- Illustrate definition with a GNN with 3 layers

- 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 $\mathcal{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$

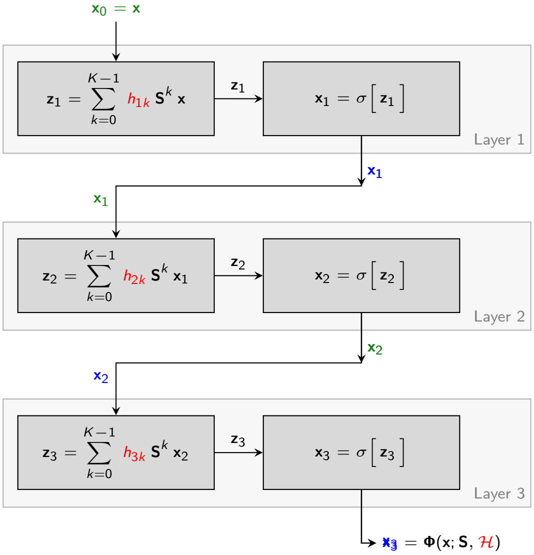


- Illustrate definition with a GNN with 3 layers
- Feed Layer 1 output as an input to Layer 2

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

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

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



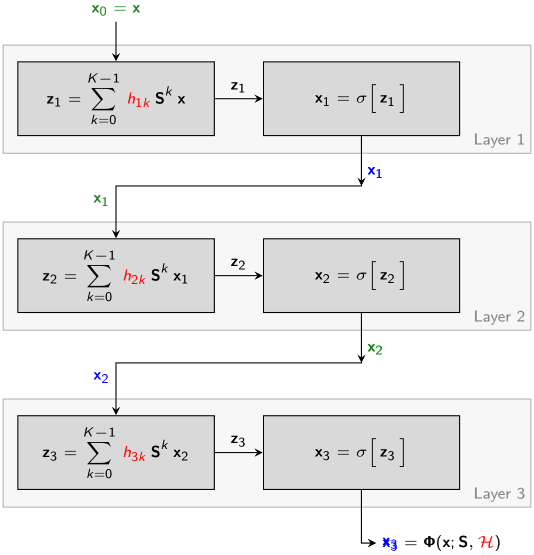
- Illustrate definition with a GNN with 3 layers

- 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 $\mathcal{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$



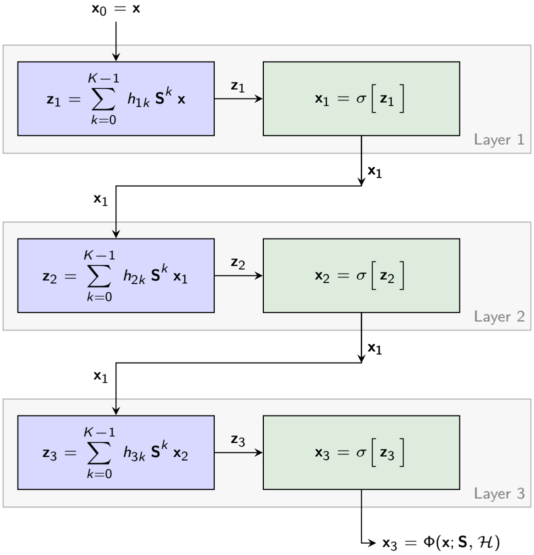
Some Observations about Graph Neural Networks

- 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

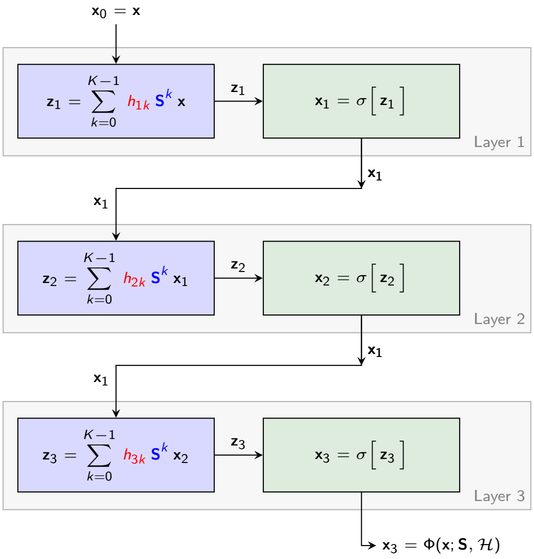


- A GNN with L layers follows L recursions of the form

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

- Filters are parametrized by...

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



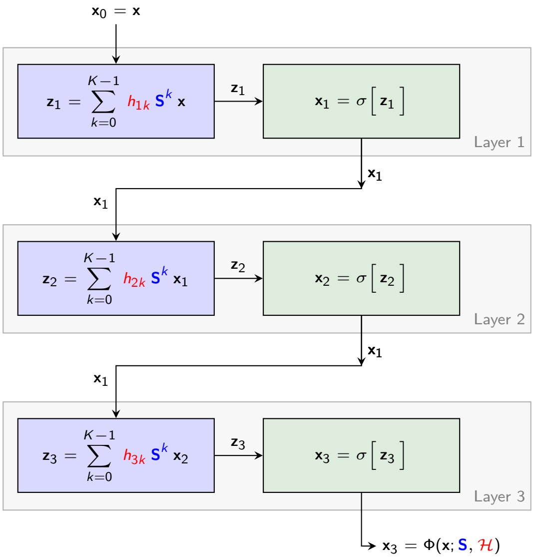
The Components ot a Graph Neural Network

- 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 $\mathcal{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_L]$

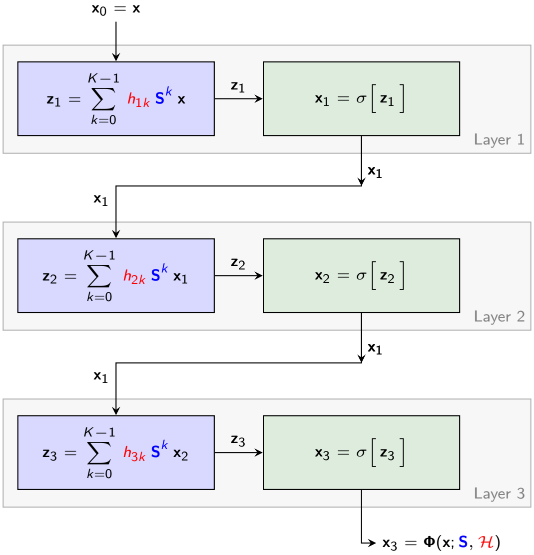


- Learn Optimal GNN tensor $\mathcal{H}^* = (\mathbf{h}_1^*, \mathbf{h}_2^*, \mathbf{h}_3^*)$ as

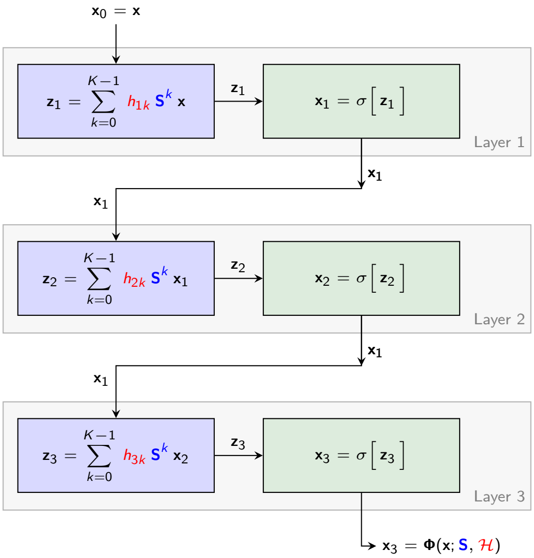
$$\mathcal{H}^* = \underset{\mathcal{H}}{\text{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\boldsymbol{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}), \mathbf{y}\Big)$$

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

$\Rightarrow$ Prior information given to the GNN

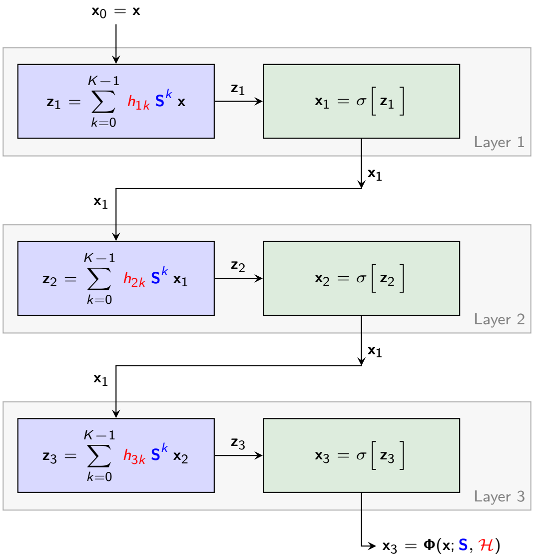


- GNNs are minor variations of graph filters
- Add pointwise nonlinearities and layer compositions
  - ⇒ Nonlinearities process individual entries
  - ⇒ Component mixing is done by graph filters only
- GNNs do work (much) better than graph filters
  - ⇒ Which is unexpected and deserves explanation
  - ⇒ 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

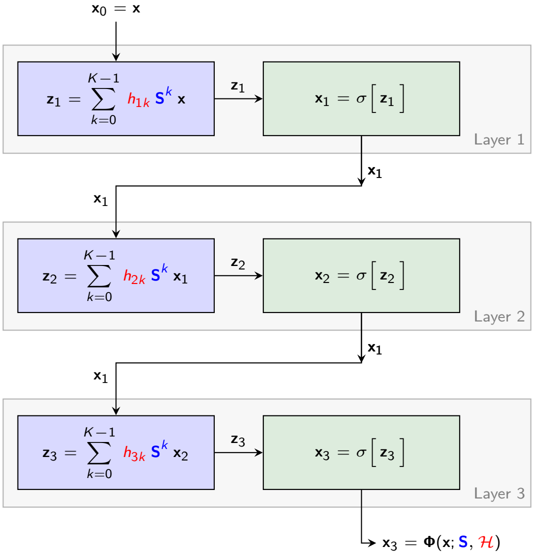
$\Rightarrow$ Enabling transference across graphs

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

$\Rightarrow$ Same as we enable transference across signals

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

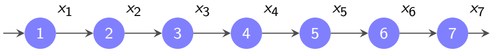
- A trained GNN is just a filter tensor $\mathcal{H}^*$



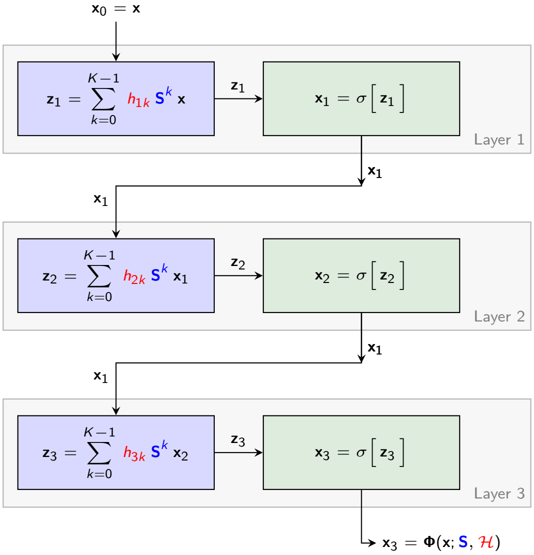
- There is no difference between CNNs and GNNs

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

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

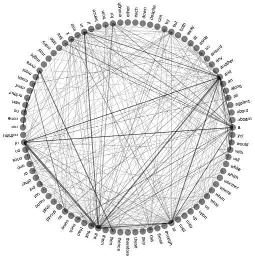


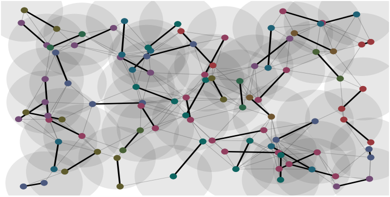
- GNNs are proper generalizations of CNNs

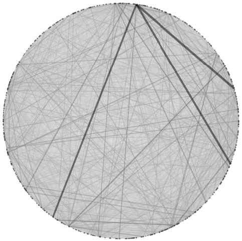


Fully Connected Neural Networks

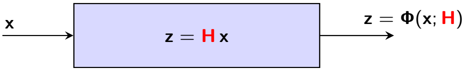
- We chose graph filters and graph neural networks (GNNs) because of our interest in graph signals
- We argued this is a good idea because they are generalizations of convolutional filters and CNNs
- We can explore this better if we go back to the road not taken $\Rightarrow$ Fully connected neural networks





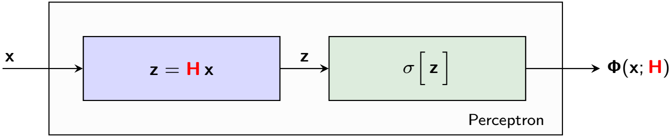


- Instead of graph filters, we choose arbitrary linear functions $\Rightarrow \mathbf{\Phi}(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x}; \mathbf{H}) = \mathbf{H}\,\mathbf{x}$



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

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



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

- 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 GNN

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

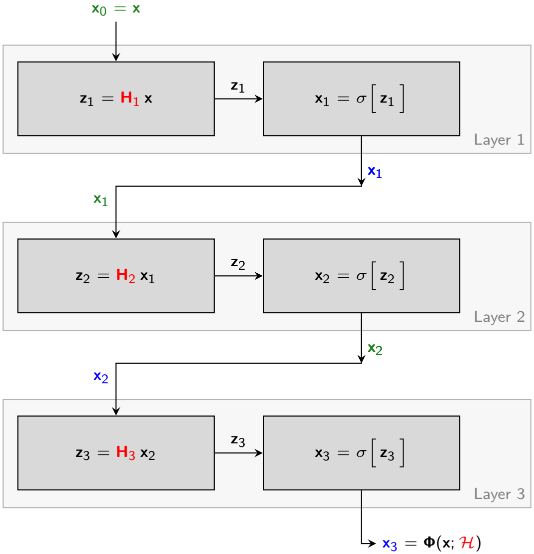
- The filter tensor $\mathcal{H} = [\mathbf{H}_1, \ldots, \mathbf{H}_L]$ is the trainable parameter.

- Illustrate definition with an FCNN with 3 layers

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

$$\mathbf{x}_1 = \sigma\Big[\,\mathbf{z}_1\,\Big] = \sigma\Big[\,\mathbf{H}_{1k}\,\mathbf{x}_0\Big]$$

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



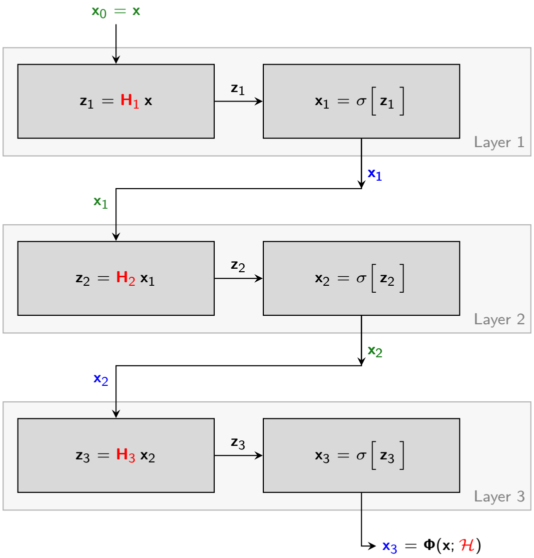
Fully Connected Neural Network Block Diagram

- Illustrate definition with an FCNN with 3 layers

- Feed Layer 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 $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$



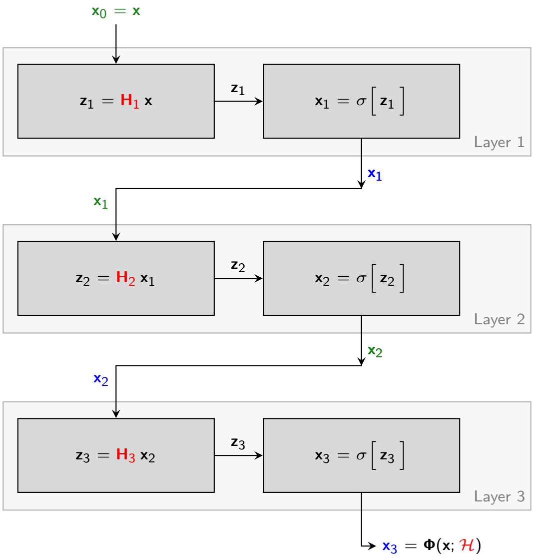
Fully Connected Neural Network Block Diagram

- 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 $\Phi(\mathbf{x};\mathcal{H})$ 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

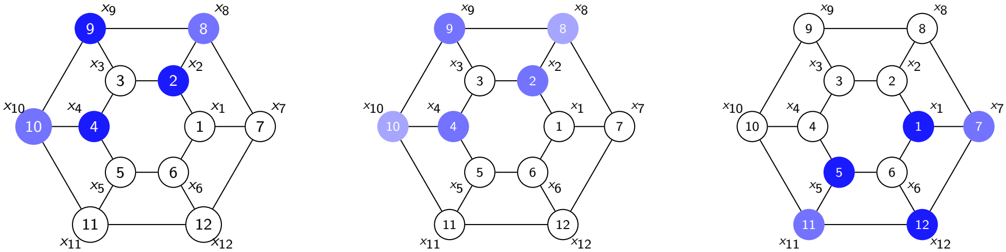
$$\min_{\mathcal{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{\Phi}(\mathbf{x};\mathcal{H}),\mathbf{y}\Big) \leq \min_{\mathcal{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H}),\mathbf{y}\Big)$$

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

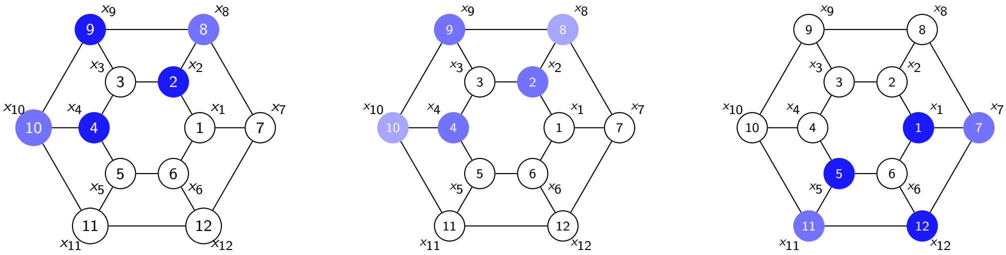
- In practice, the GNN does better because it generalizes better to unseen signals

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

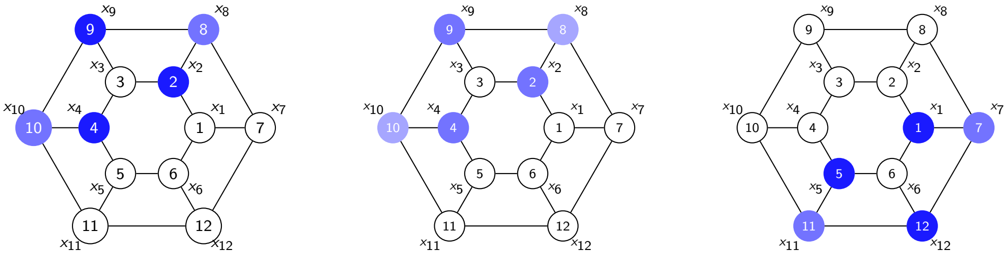
- Suppose the graph represents a recommendation system where we want to fill empty ratings
- We observe ratings with the structure in the left. But we do not observe examples like the other two
- From examples like the one in the left, the NN learns how to fill the middle signal but not the right



- 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

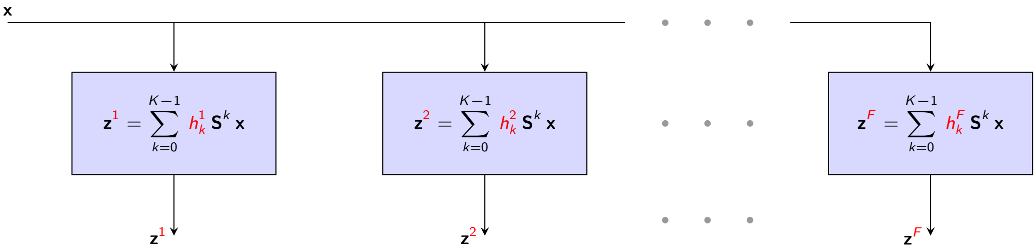


Graph Filter Banks

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

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

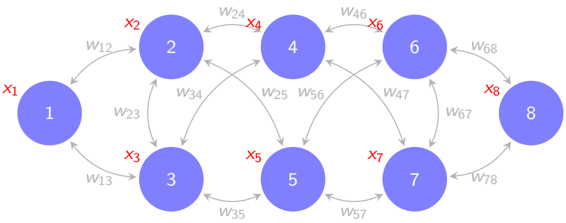
- Filter f in the bank uses coefficients $\mathbf{h}^f = [h_1^f; \ldots; h_{K-1}^f]$ $\Rightarrow$ Output $\mathbf{z}^f$ is a graph signal



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

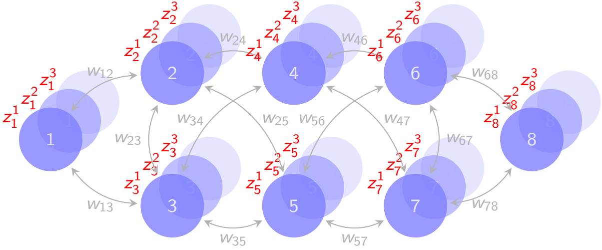
Filter Bank Outputs: Multiple Features

- 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 $\mathbf{z}_i$



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

- 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 $\mathbf{z}_i$

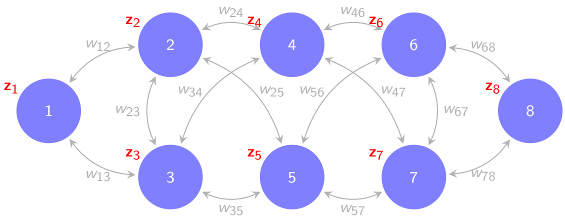


$$\mathbf{Z} = \begin{bmatrix} z_1^1 & \cdots & z_1^f & \cdots & z_1^F \\ \vdots & & \vdots & & \vdots \\ z_i^1 & \cdots & z_i^f & \cdots & z_i^F \\ \vdots & & \vdots & & \vdots \\ z_n^1 & \cdots & z_n^f & \cdots & z_n^F \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_i \\ \vdots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

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

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 $\mathbf{z}_i$



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

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

Theorem (Output Energy of a Graph Filter)

Consider graph filter $\mathbf{h}$ with coefficients h_k and frequency response $\tilde{h}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$. The energy of the filter's output $\mathbf{z} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$ is given by

$$\|\,\mathbf{z}\,\|^2 = \sum_{i=1}^{n} \Big(\,\tilde{h}(\lambda_i)\,\tilde{x}_i\,\Big)^2$$

where λ_i are eigenvalues of symmetric $\mathbf{S}$ and $\tilde{x}_i$ are components of the GFT of $\mathbf{x}$, $\tilde{\mathbf{x}} = \mathbf{V}^H\mathbf{x}$ is

Proof: The GFT is a unitary transform that preserves energy. Indeed, with $\tilde{\mathbf{z}} = \mathbf{V}^H\mathbf{z}$ we have

$$\left\| \tilde{\mathbf{z}} \right\|^2 = \tilde{\mathbf{z}}^H\tilde{\mathbf{z}} = \left(\mathbf{V}^H\mathbf{z}\right)^H\left(\mathbf{V}^H\mathbf{z}\right) = \mathbf{z}^H\mathbf{V}\mathbf{V}^H\mathbf{z} = \mathbf{z}^H\,\mathbf{I}\,\mathbf{z} = \left\| \mathbf{z} \right\|^2$$

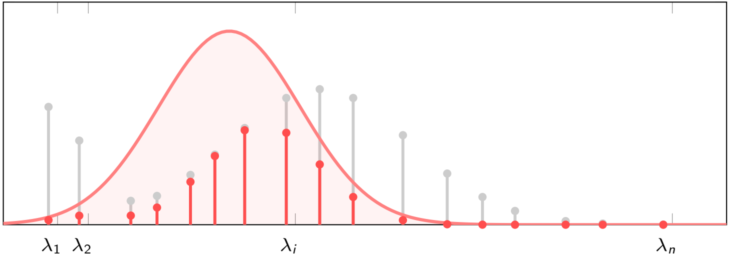
- We know that graph filters are pointwise in the frequency domain $\Rightarrow \tilde{z}_i = \tilde{h}(\lambda_i)\tilde{x}_i$

$$\left\| \tilde{\mathbf{z}} \right\|^2 = \tilde{\mathbf{z}}^H\tilde{\mathbf{z}} = \sum_{i=1}^{n} \tilde{z}_i^2 = \sum_{i=1}^{n} \left(\tilde{h}^f(\lambda_i)\,\tilde{x}_i \right)^2$$

- We have the energy expressed in the form we want. Except that it is in the frequency domain.

- But we have just seen the GFT preserves energy $\Rightarrow \left\| \mathbf{z} \right\|^2 = \left\| \tilde{\mathbf{z}} \right\|^2 = \sum_{i=1}^{n} \left(\tilde{h}(\lambda_i)\,\tilde{x}_i \right)^2$ ■

- The energy that graph filters let pass is a sort of "area under the frequency response curve."
- Graph Filter banks are helpful in identifying frequency signatures of different signals

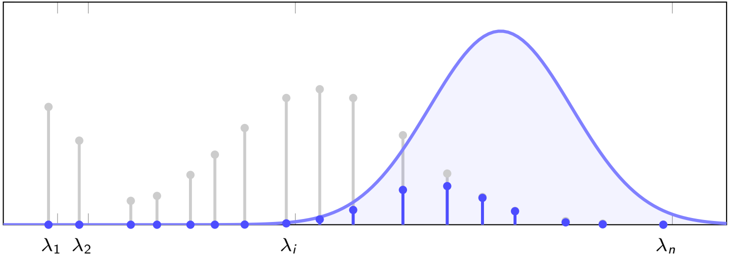


- Filter banks scatter the energy of signal $\mathbf{x}$ into the signals $\mathbf{z}^f$ at the output of the filters.

  $\Rightarrow$ Different signals concentrate energy on different outputs $\mathbf{z}^f$

Filter Banks in the Graph Frequency Domain

- The energy that graph filters let pass is a sort of "area under the frequency response curve."
- Graph Filter banks are helpful in identifying frequency signatures of different signals

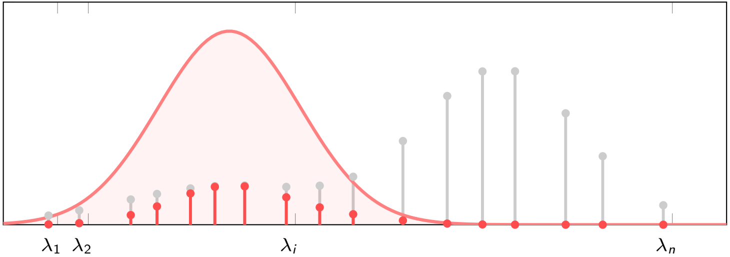


- Filter banks scatter the energy of signal $\mathbf{x}$ into the signals $\mathbf{z}^f$ at the output of the filters.

  $\Rightarrow$ Different signals concentrate energy on different outputs $\mathbf{z}^f$

Filter Banks in the Graph Frequency Domain

- The energy that graph filters let pass is a sort of "area under the frequency response curve."
- Graph Filter banks are helpful in identifying frequency signatures of different signals

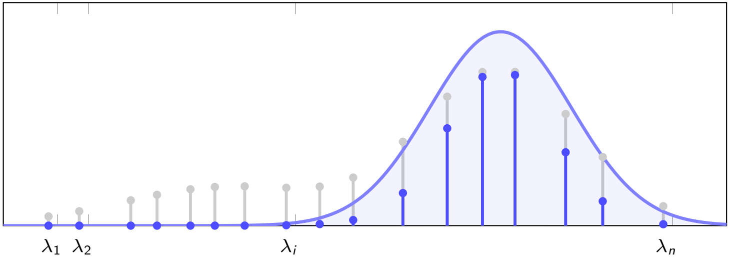


- Filter banks scatter the energy of signal $\mathbf{x}$ into the signals $\mathbf{z}^f$ at the output of the filters.

$\Rightarrow$ Different signals concentrate energy on different outputs $\mathbf{z}^f$

Filter Banks in the Graph Frequency Domain

- The energy that graph filters let pass is a sort of "area under the frequency response curve."
- Graph Filter banks are helpful in identifying frequency signatures of different signals

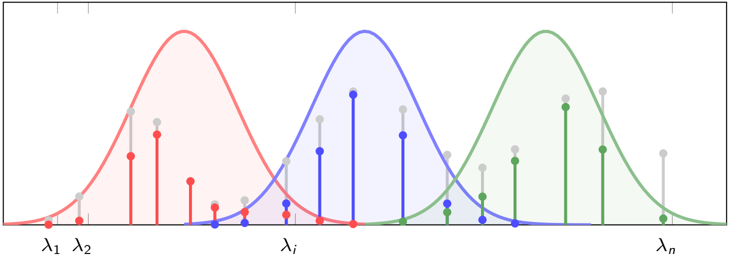


- Filter banks scatter the energy of signal $\mathbf{x}$ into the signals $\mathbf{z}^f$ at the output of the filters.

$\Rightarrow$ Different signals concentrate energy on different outputs $\mathbf{z}^f$

- The filter bank isolates groups of frequency components

$\Rightarrow$ Energy of bank output $\mathbf{z}^f = \sum_{k=0}^{\infty} h_k^f \mathbf{S}^k \mathbf{x}$ is area under the curve $\Rightarrow \| \mathbf{z}^f \|^2 = \sum_{i=1}^{n} \left(\tilde{h}^f(\lambda_i) \, \tilde{x}_i \right)^2$



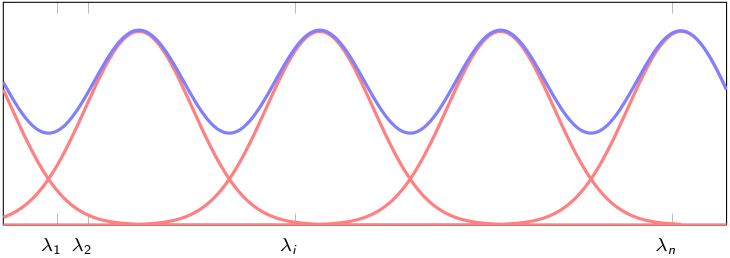
- We use the filter bank to identify signals with different spectral signatures.

- The GFT preserves energy $\Rightarrow$ It scatters information. But it doesn't loose information
- A filter bank is a frame if there exist constants $m \leq M$ $\Rightarrow m \left\|\mathbf{x}\right\|^2 \leq \sum_{f=1}^{F} \left\|\mathbf{z}^f\right\|^2 \leq M \left\|\mathbf{x}\right\|^2$
- A filter banks is a tight frame if $m = M = 1$ $\Rightarrow \left\|\mathbf{x}\right\|^2 = \sum_{f=1}^{F} \left\|\mathbf{z}^f\right\|^2$
- No signal is vanquished by a frame. Energy is preserved by a tight frame

Frames in the Graph Frequency Domain

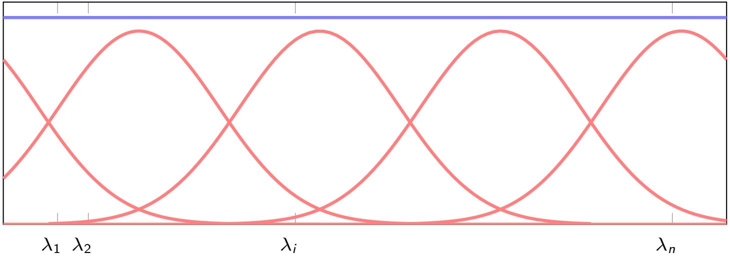
- Because filters are pointwise in the GFT domain, a frame must satisfy $\Rightarrow m \leq \sum_{f=1}^{F} \left[\tilde{h}^f(\lambda) \right]^2 \leq M$

- All frequencies λ must have at least one filter $\mathbf{h}^f$ with response $m \leq \left[\tilde{h}^f(\lambda) \right]^2$



- Likewise, a tight frame must be such that for all $\lambda \;\Rightarrow \sum_{f=1}^{F} \Big[\, \tilde{h}^f(\lambda) \Big]^2 = 1$

- A Sufficient condition is that all frequencies accumulate unit energy when summing across all filters



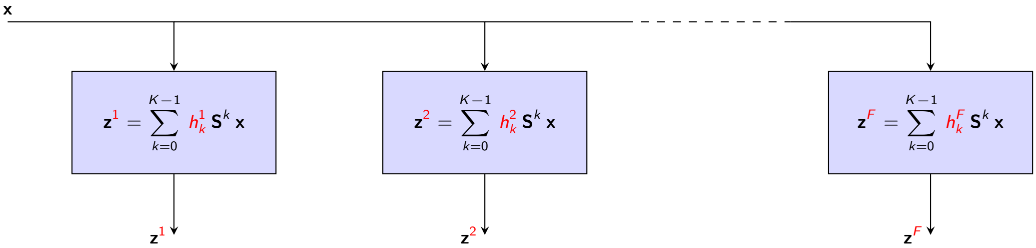
- We will not design filter banks. We will learn them. But keeping them close to frames is good.

Multiple Feature GNNs

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

- Filter banks output a collection of multiple graph signals $\Rightarrow$ A matrix graph signal $\mathbf{Z} = [\mathbf{z}^1, \ldots, \mathbf{z}^F]$

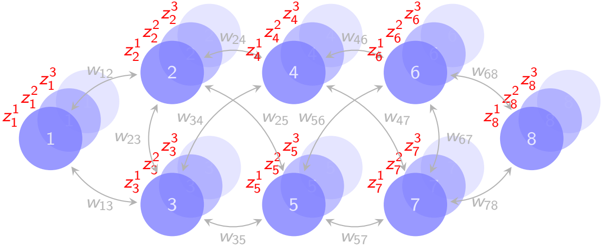
- The F graph signals $\mathbf{z}^f$ represent F features per node. A vector $\mathbf{z}_i$ supported at each node



- 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 $\mathbf{Z} = [\mathbf{z}^1, \ldots, \mathbf{z}^F]$

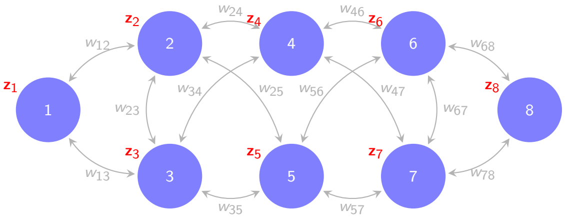
- The F graph signals $\mathbf{z}^f$ represent F features per node. A vector $\mathbf{z}_i$ supported at each node



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

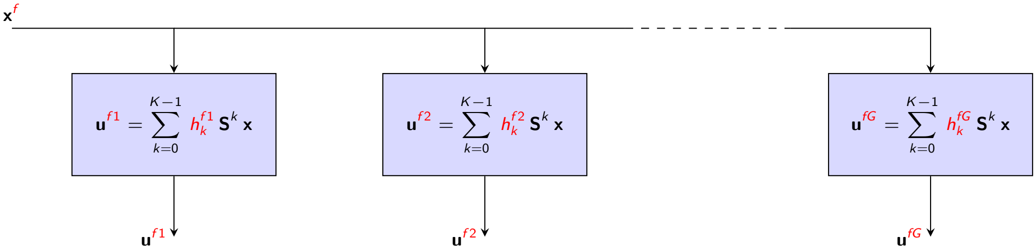
Multiple Feature (Matrix) Graph Signals

- 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

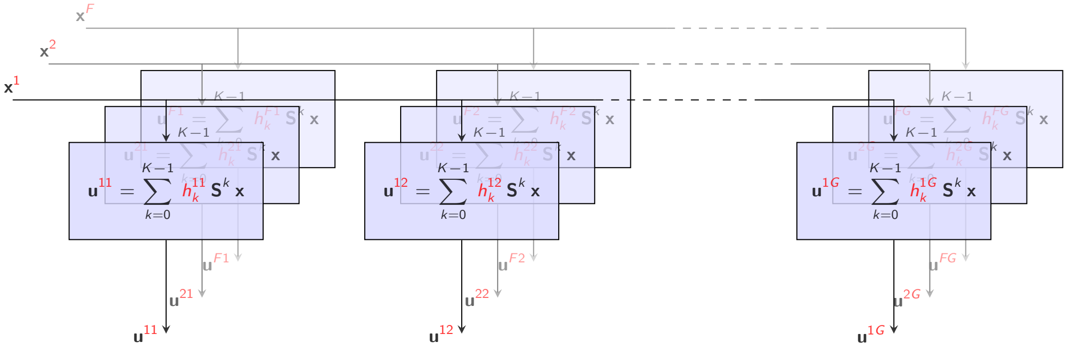


- We would now like to process multiple feature graph signals. Process each feature with a filterbank.

- 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

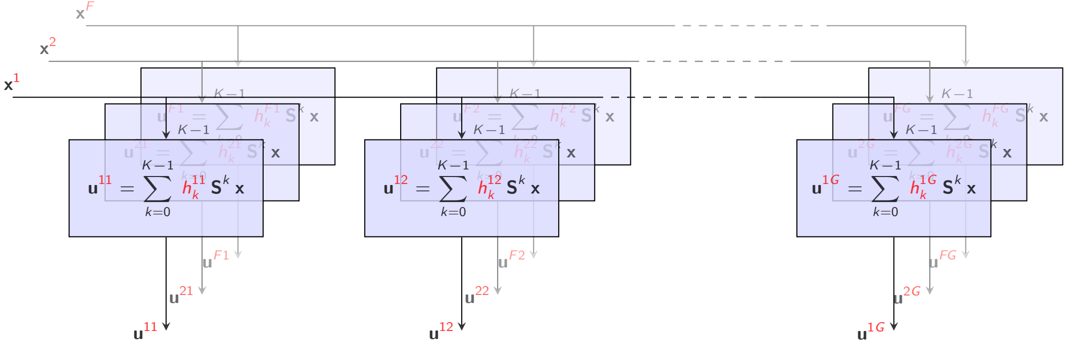


$\mathbf{z}^1 = \mathbf{u}^{11} + \mathbf{u}^{21} + \ldots + \mathbf{u}^{F1}$

$\mathbf{z}^2 = \mathbf{u}^{12} + \mathbf{u}^{22} + \ldots + \mathbf{u}^{F2}$

$\mathbf{z}^2 = \mathbf{u}^{1G} + \mathbf{u}^{2G} + \ldots + \mathbf{u}^{FG}$

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



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

- 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

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

- 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

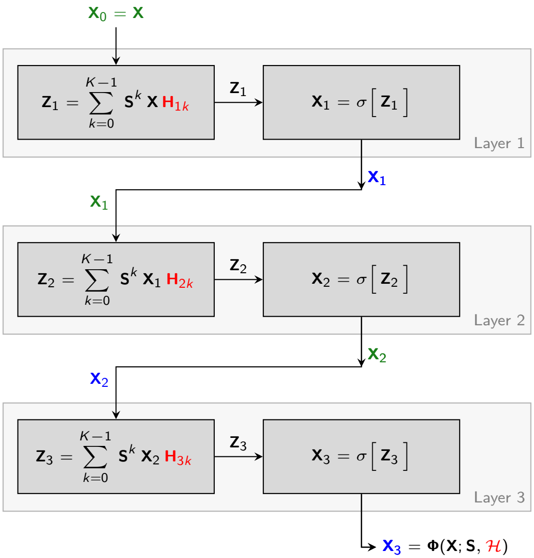
MIMO GNN Block Diagram

- 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

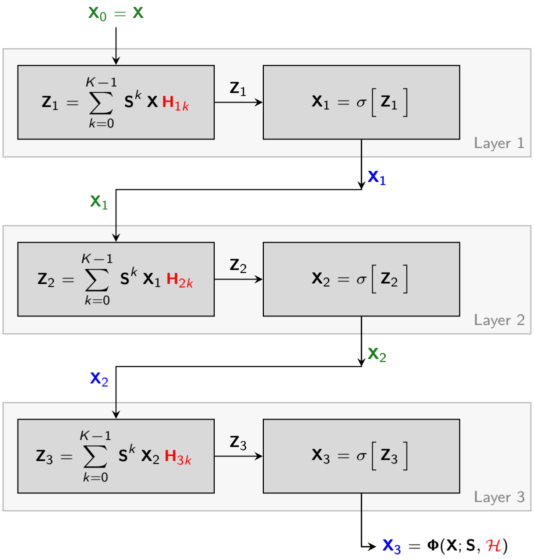
- We illustrate with a MIMO GNN with 3 layers

- 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 $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$



- We illustrate with a MIMO GNN with 3 layers
- Feed Layer 2 output (F_2 features) as an input to Layer 3

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

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

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