

Graph Neural Networks

Gonzalo Mateos

Dept. of ECE and Goergen Institute for Data Science

University of Rochester

gmateosb@ece.rochester.edu

<http://www.hajim.rochester.edu/ece/sites/gmateos/>

Slides credits: Prof. A. Ribeiro and his group at Penn

April 24, 2023

- (I) The **why** \Rightarrow Graphs appear in scores of settings \Rightarrow They are **pervasive models of structure**

- (II) The **how** \Rightarrow We should use a **neural network** \Rightarrow Fully connected neural networks do not scale
 \Rightarrow **Convolutions** (in time or graphs) **are the key to scalable** machine learning

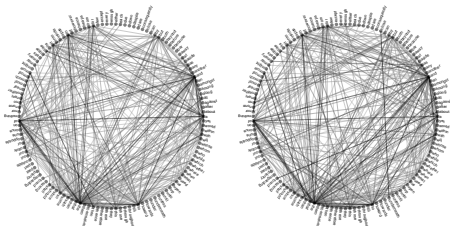
- (III) **Convolutional filters** in Euclidean space and convolutional filters on graphs

- (IV) **Convolutional neural networks** and Graph (convolutional) neural networks

Machine Learning on Graphs: The Why

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

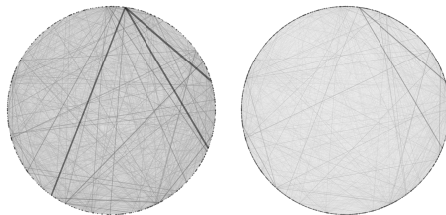
Authorship Attribution



Identify the author of a text of unknown provenance

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

Recommendation Systems

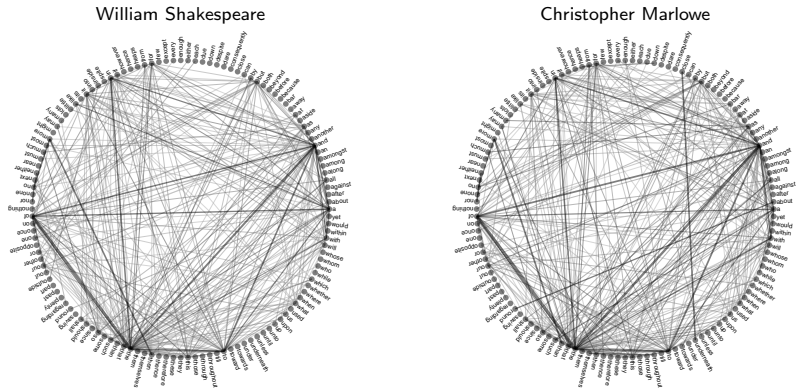


Predict the rating a customer would give to a product

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

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

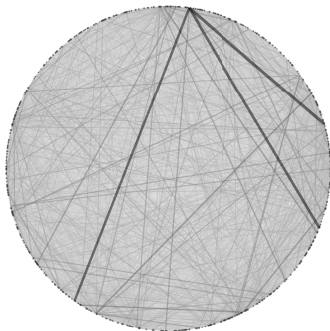
- ▶ **Nodes** represent different **function words** and **edges** how often words appear close to each other
⇒ A proxy for the different ways in which different authors use the English language grammar



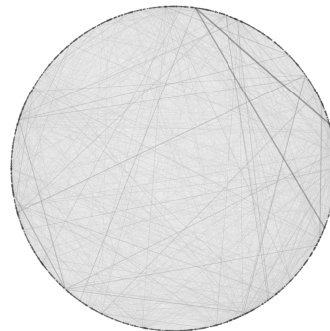
- ▶ WAN differences **differentiate the writing styles of Marlowe and Shakespeare** in, e.g., Henry VI

- ▶ **Nodes** represent different **customers** and **edges** their average **similarity in product ratings**
 - ⇒ The graph informs the completion of ratings when some are unknown and are to be predicted

Variation Diagram for Original (sampled) ratings



Variation Diagram for Reconstructed (predicted) ratings



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

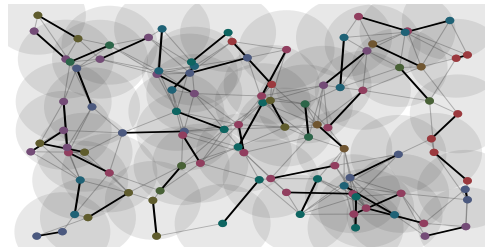
- ▶ Graphs are **more than data structures** \Rightarrow They are models of **physical systems with multiple agents**

Decentralized Control of Autonomous Systems

Coordinate a team of agents without central coordination

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

Wireless Communications Networks



Manage interference when allocating bandwidth and power

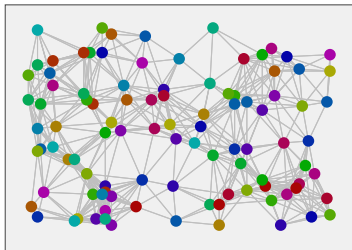
Eisen-Ribeiro '19, arxiv.org/abs/1909.01865

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

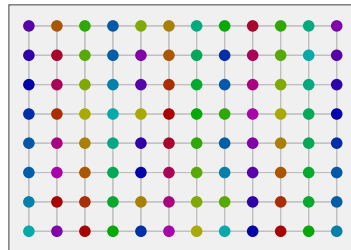
Machine Learning on Graphs: The How

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

Challenge is we want to run a NN over this



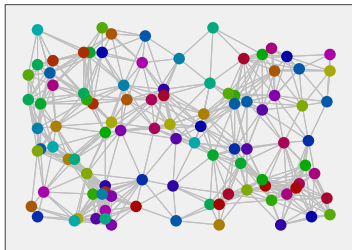
But we are good at running NNs over this



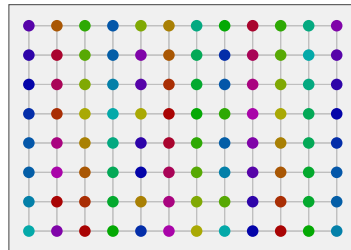
- ▶ Generic NNs do not scale to large dimensions \Rightarrow **Convolutional Neural Networks (CNNs)** do scale

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

Process graphs with **graph convolutional** NNs



Process images with **convolutional** NNs

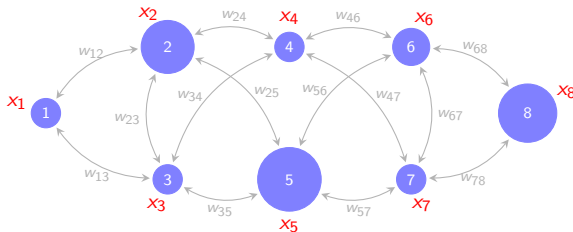


- ▶ **Generalize convolutions to graphs** \Rightarrow Compose graph filter banks with **pointwise nonlinearities**
- ▶ Stack in **layers** to create a **graph (convolutional) Neural Network (GNN)**

Graph Signals

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

- ▶ Consider a given graph \mathcal{G} with N_V nodes and shift operator \mathbf{S} (e.g., adjacency or Laplacian matrices)
- ▶ A graph signal is a vector $\mathbf{x} \in \mathbb{R}^{N_V}$ 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** $\Rightarrow (\mathbf{S}, \mathbf{x})$



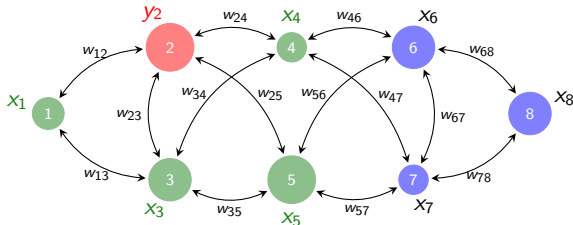
- ▶ The graph is an **expectation of proximity or similarity** between components of the signal \mathbf{x}

► 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 $y_i = \sum_{j \in \mathcal{N}_i} w_{ij} x_j = \sum_j w_{ij} x_j$

\Rightarrow Stronger weights contribute more to the diffusion output

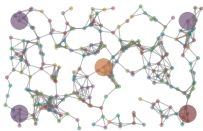
\Rightarrow Codifies a **local operation** where components are mixed with components of **neighboring nodes**.



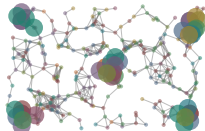
- ▶ **Compose** the diffusion operator to produce **diffusion sequence** \Rightarrow defined recursively as

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

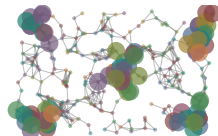
- ▶ Can **unroll** the recursion and write the diffusion sequence as the **power sequence** $\Rightarrow \mathbf{x}^{(k)} = \mathbf{S}^k \mathbf{x}$



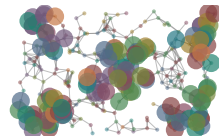
$$\mathbf{x}^{(0)} = \mathbf{x} = \mathbf{S}^0 \mathbf{x}$$



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

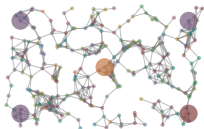


$$\mathbf{x}^{(2)} = \mathbf{S}\mathbf{x}^{(1)} = \mathbf{S}^2 \mathbf{x}$$

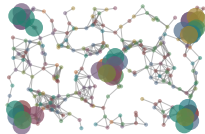


$$\mathbf{x}^{(3)} = \mathbf{S}\mathbf{x}^{(2)} = \mathbf{S}^3 \mathbf{x}$$

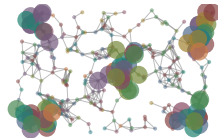
- ▶ The k th element of the diffusion sequence $x^{(k)}$ diffuses information to k -hop neighborhoods
⇒ One reason why we use the diffusion sequence to define graph convolutions
- ▶ We have two definitions. One recursive. The other one using powers of S
⇒ Always implement the recursive version. The power version is good for analysis



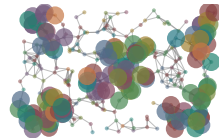
$$x^{(0)} = x = S^0 x$$



$$x^{(1)} = Sx^{(0)} = S^1 x$$



$$x^{(2)} = Sx^{(1)} = S^2 x$$



$$x^{(3)} = Sx^{(2)} = S^3 x$$

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}

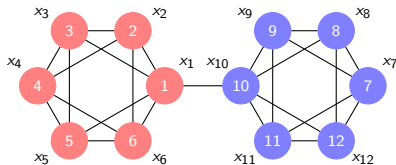
$$\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{\infty} h_k \mathbf{S}^k$$

- ▶ 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}

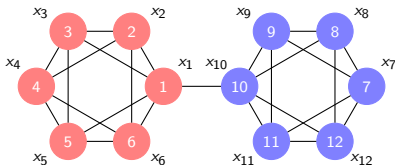
- ▶ Graph convolutions aggregate information growing from local to global neighborhoods
- ▶ Consider a signal \mathbf{x} supported on a graph with **shift operator \mathbf{S}** . Along with **filter $\mathbf{h} = \{h_k\}_{k=0}^{K-1}$**



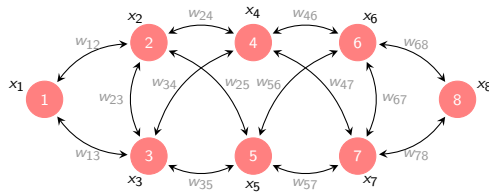
- ▶ Graph convolution output $\Rightarrow \mathbf{y} = \mathbf{h} \star_{\mathbf{S}} \mathbf{x} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \dots = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$

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

Graph Filter on a Graph

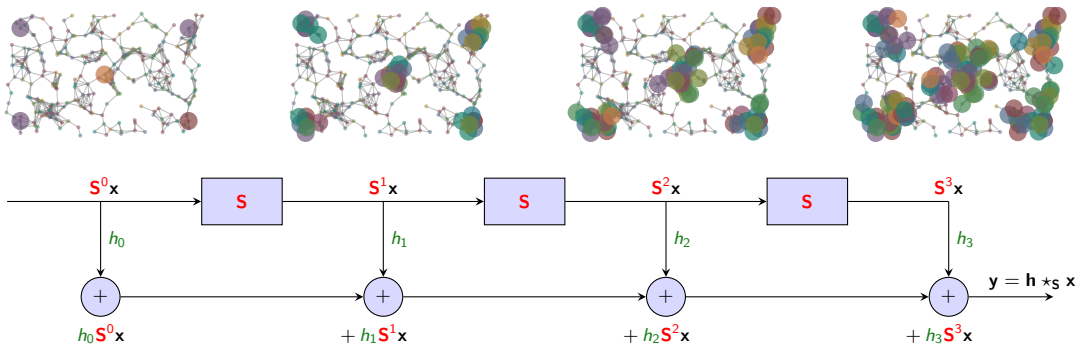


Same Graph Filter on Another Graph



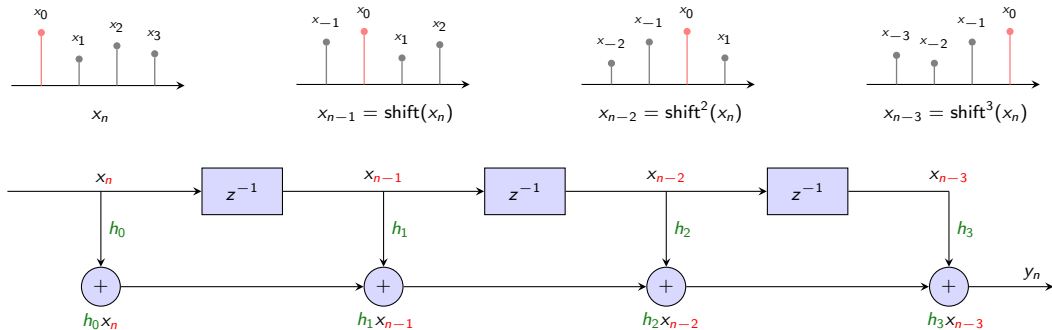
- ▶ Graph convolution output $\Rightarrow \mathbf{y} = \mathbf{h} \star_{\mathbf{S}} \mathbf{x} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \dots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$
- ▶ Output depends on the 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**



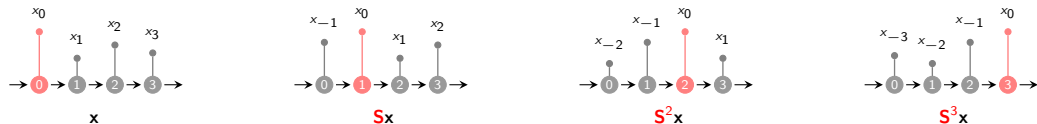
Time Convolutions as a Particular Case of Graph Convolutions

- **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})**



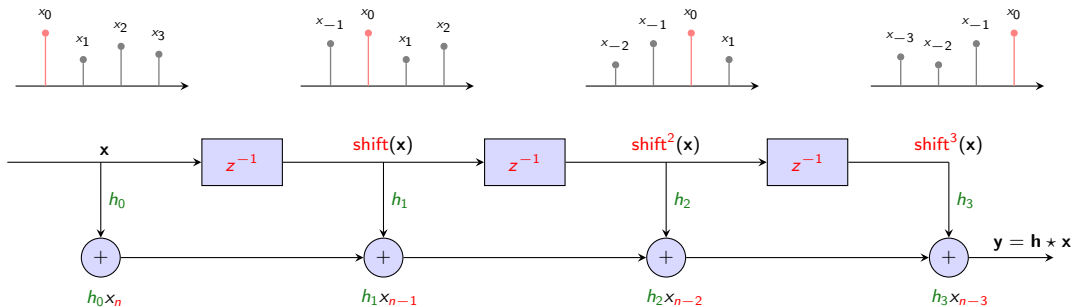
- Time shift is reinterpreted as **multiplication by the adjacency matrix \mathbf{S}** of the line graph

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

- 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 as a Polynomial on the Line Adjacency

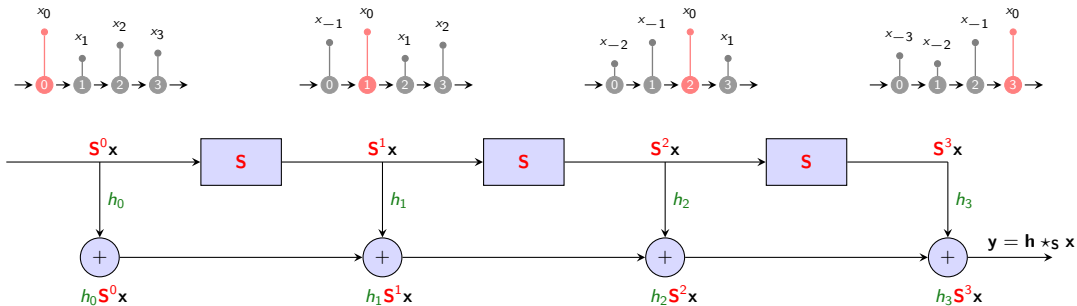
- ▶ The convolution operation is a linear combination of **shifted** versions of the input signal
- ▶ But we now know that time shifts are **multiplications with the adjacency matrix S** of line graph



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

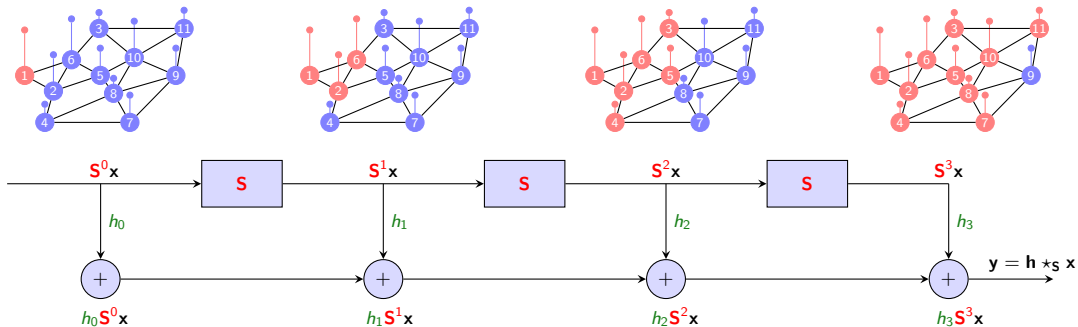
The Convolution as a Polynomial on the Line Adjacency

- ▶ 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 $\Rightarrow y = h * x = \sum_{k=0}^{K-1} h_k S^k x$

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



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

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

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

▶ 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 $\mathbf{V}^H \mathbf{V}$

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



- ▶ In the graph frequency domain graph filters are **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** $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$, the graph frequency response is the polynomial

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

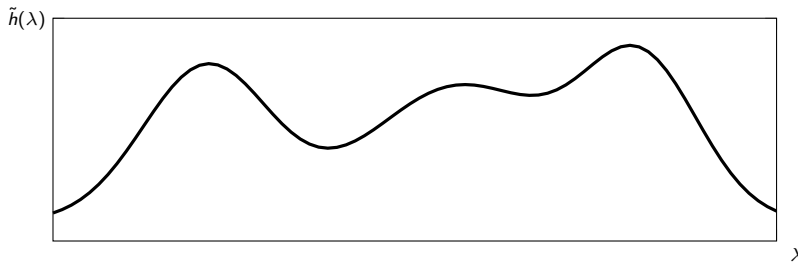
Definition (Frequency Response of a Graph Filter)

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

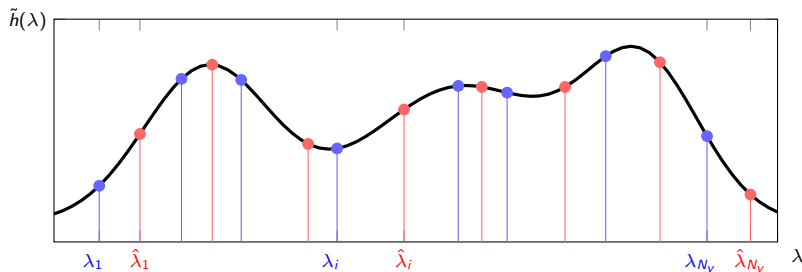
$$\tilde{h}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$$

- ▶ 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 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 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{N_v}$

\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 $\ell(\mathbf{y}, \Phi(\mathbf{x}))$ averaged over training set

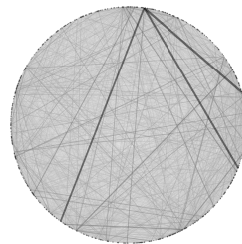
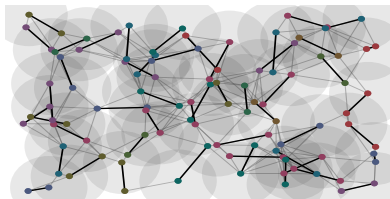
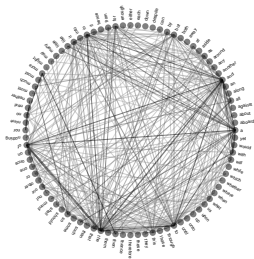
$$\Phi^* = \operatorname{argmin}_{\Phi \in \mathcal{C}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}} \ell(\mathbf{y}, \Phi(\mathbf{x}))$$

▶ 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

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

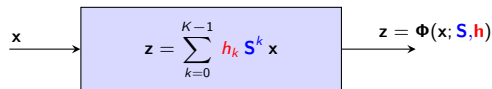
$$\Phi^* = \underset{\Phi \in \mathcal{C}}{\operatorname{argmin}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}} \ell(\mathbf{y}, \Phi(\mathbf{x}))$$

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



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

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

Graph Neural Networks (GNNs)

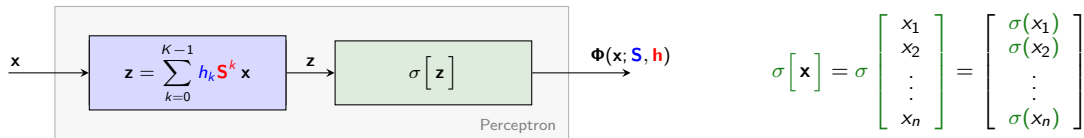
- ▶ A **pointwise nonlinearity** is a nonlinear function applied componentwise. **Without mixing entries**

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

- ▶ 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** $\Rightarrow \Phi(\mathbf{x}) = \sigma \left[\sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x} \right] = \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h})$



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

\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** $\mathbf{h}_1 = [h_{10}, \dots, h_{1,K-1}]$ to produce **output** \mathbf{x}_1

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

- ▶ 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** $\mathbf{h}_2 = [h_{20}, \dots, h_{2,K-1}]$ to produce **output** \mathbf{x}_2

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

- ▶ 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 $\mathbf{h}_\ell = [h_{\ell 0}, \dots, h_{\ell, K-1}]$** to produce **output \mathbf{x}_ℓ**

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

- ▶ 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** $\Rightarrow \mathbf{x}_L = \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h}_1, \dots, \mathbf{h}_L) = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$
- ▶ The **filter tensor $\mathcal{H} = [\mathbf{h}_1, \dots, \mathbf{h}_L]$** is the trainable parameter. The graph shift is prior information

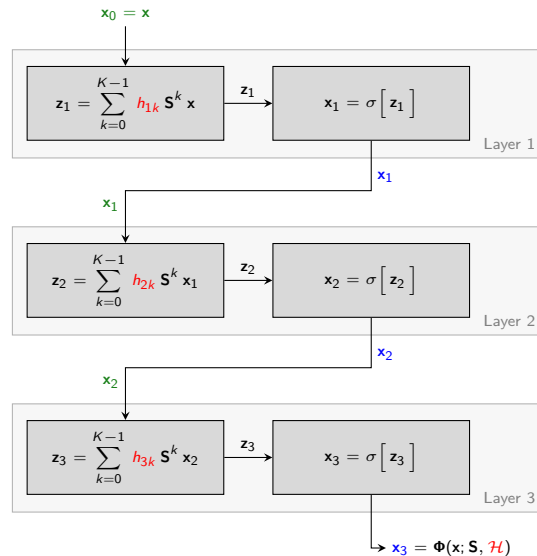
- ▶ Illustrate definition with a GNN with 3 layers

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

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

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

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



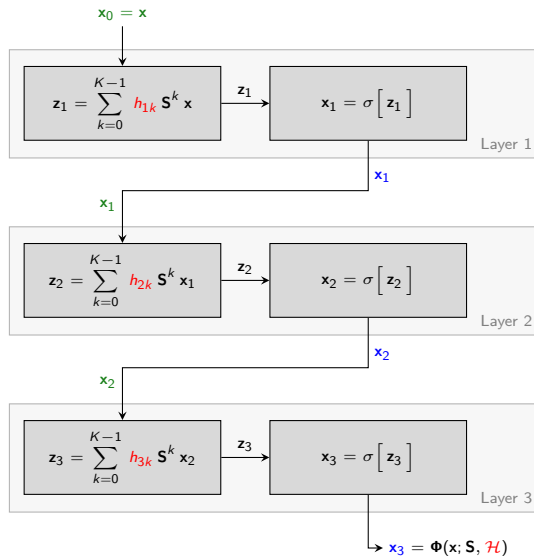
- ▶ Illustrate definition with a GNN with 3 layers

- ▶ Feed Layer 1 output as an input to Layer 2

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

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

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



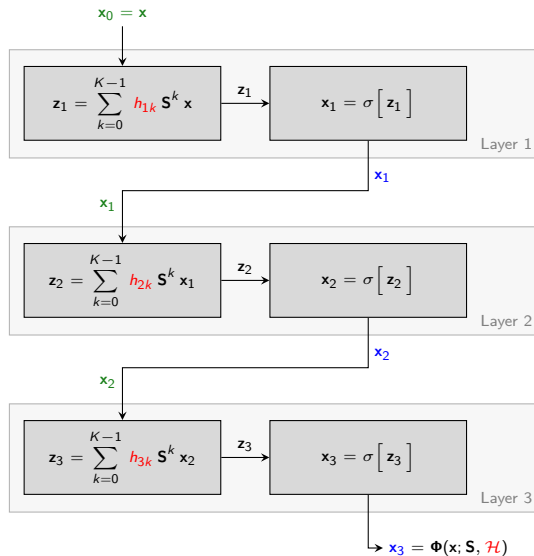
- ▶ Illustrate definition with a GNN with 3 layers

- ▶ Feed Layer 2 output as an input to Layer 3

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

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

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



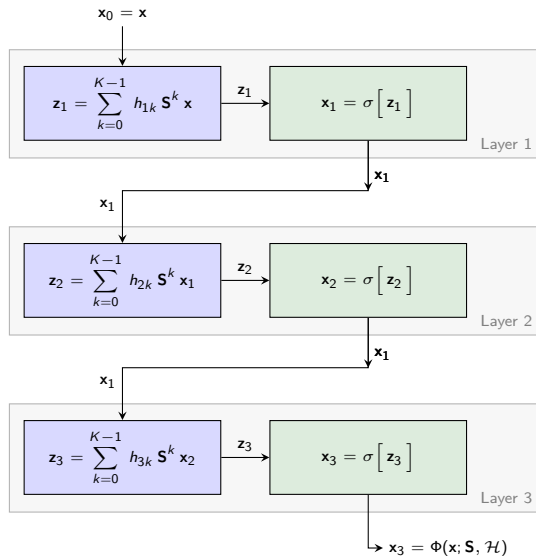
Some Observations about Graph Neural Networks

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

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

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

⇒ Compositions of **Filters** & **Pointwise nonlinearities**



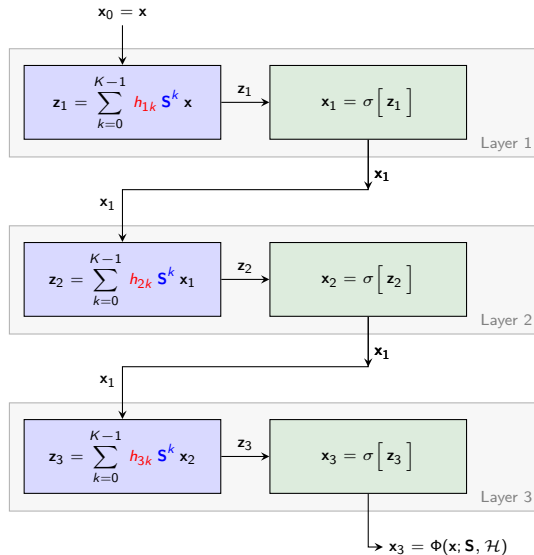
The Components of a Graph Neural Network

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

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

- ▶ Filters are parametrized by...

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

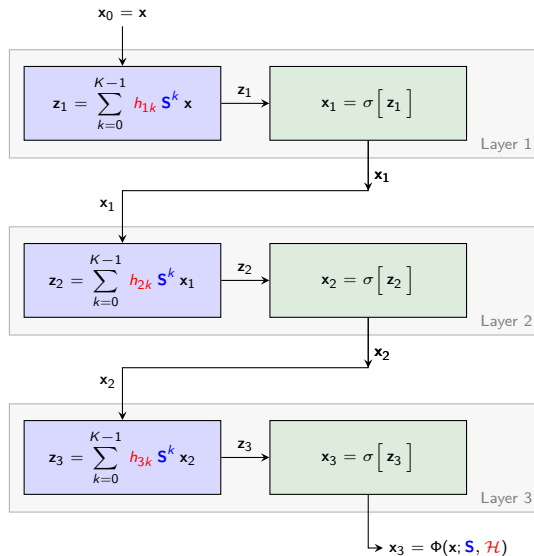


- ▶ A GNN with L layers follows L recursions of the form

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

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

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

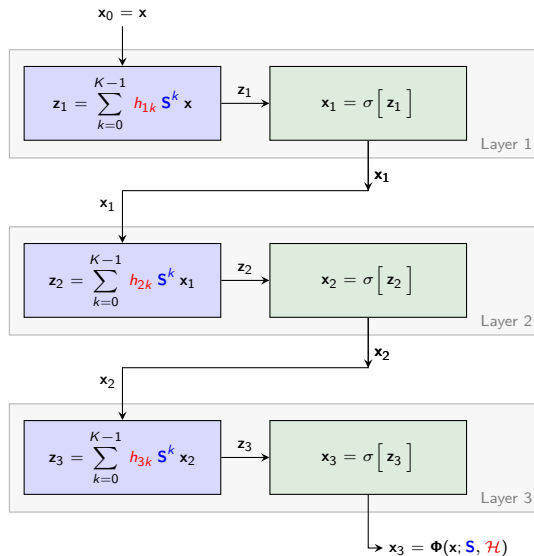


- ▶ Learn Optimal GNN tensor $\mathcal{H}^* = (\mathbf{h}_1^*, \mathbf{h}_2^*, \mathbf{h}_3^*)$ as

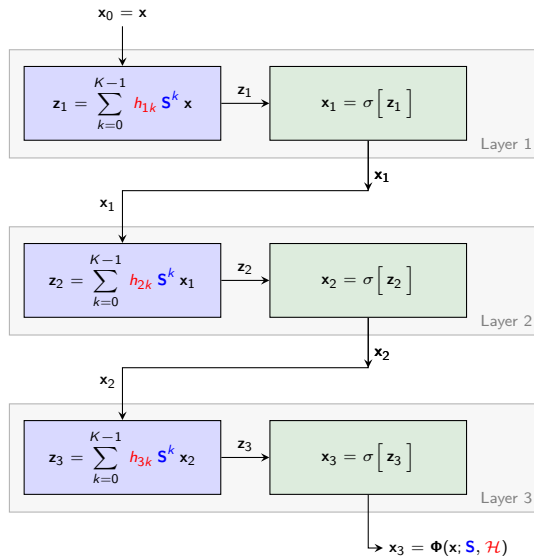
$$\mathcal{H}^* = \operatorname{argmin}_{\mathcal{H}} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}} \ell(\Phi(\mathbf{x}; \mathbf{S}, \mathcal{H}), \mathbf{y})$$

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

⇒ Prior information given to the GNN



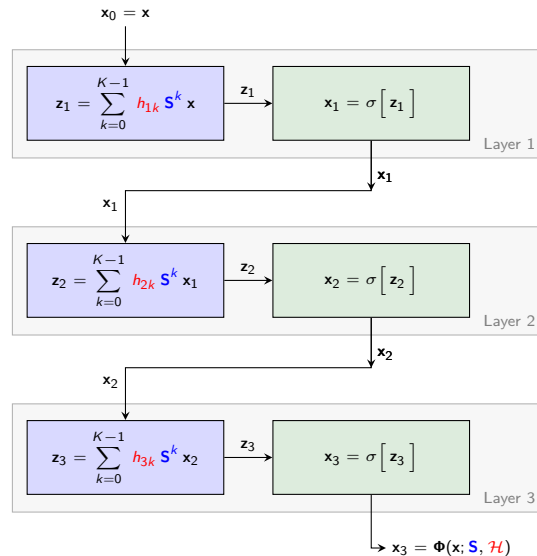
- ▶ GNNs are **minor variations** of graph filters
- ▶ Add **pointwise** nonlinearities and layer **compositions**
 - ⇒ Nonlinearities process individual entries
 - ⇒ Component mixing is done by graph filters only
- ▶ **GNNs do work** (much) **better** than graph filters
 - ⇒ Which is **unexpected**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



► GNN Output depends on the graph S

► Interpret S as a parameter

⇒ Encodes prior information. As we have done so far



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

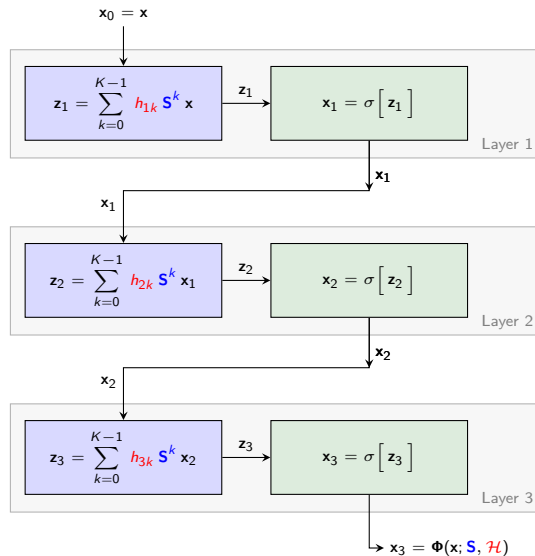
⇒ Enabling **transference across graphs**

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

⇒ Same as we enable **transference across signals**

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

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

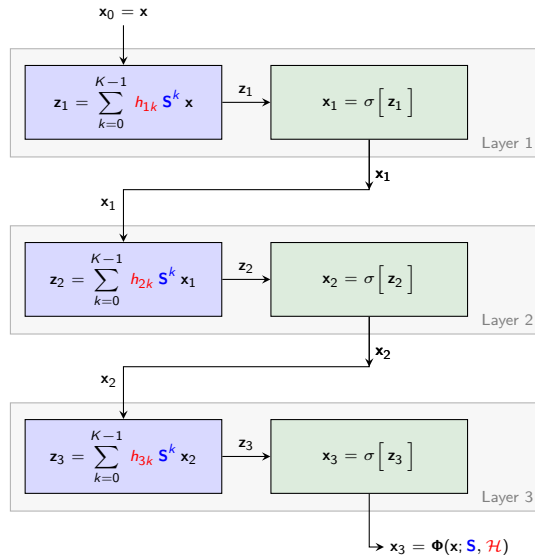


- ▶ There is **no difference** between CNNs and GNNs
- ▶ To recover a CNN just **particularize** the **shift** operator the **adjacency** matrix of the **directed line graph**

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

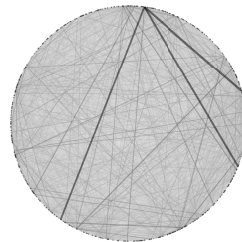
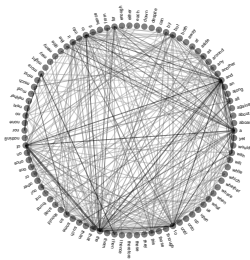


- ▶ GNNs are proper generalizations of CNNs

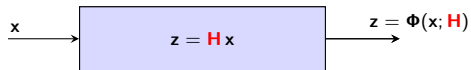


Fully Connected Neural Networks

- ▶ 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**

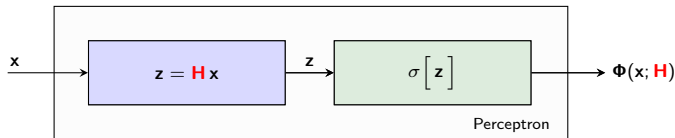


- ▶ Instead of graph filters, we choose **arbitrary linear functions** $\Rightarrow \Phi(\mathbf{x}) = \Phi(\mathbf{x}; \mathbf{H}) = \mathbf{H} \mathbf{x}$



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

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



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

- ▶ 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}_ℓ** to produce **output \mathbf{x}_ℓ**

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

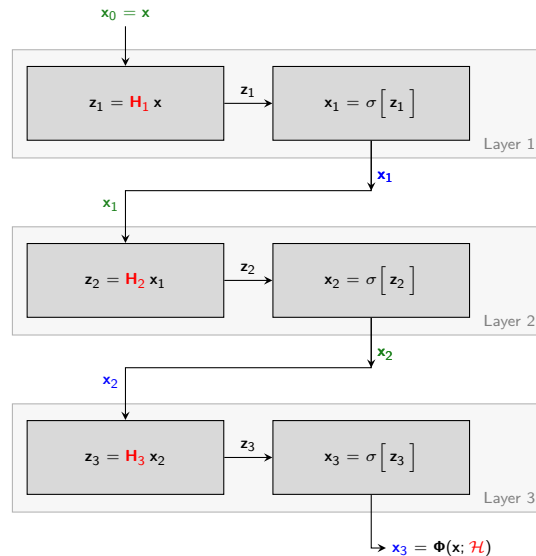
- ▶ With the convention that the **Layer 1 input** is **$\mathbf{x}_0 = \mathbf{x}$** , this provides a **recursive definition of a FCNN**
- ▶ If it has L layers, the **FCNN output** $\Rightarrow \mathbf{x}_L = \Phi(\mathbf{x}; \mathbf{H}_1, \dots, \mathbf{H}_L) = \Phi(\mathbf{x}; \mathcal{H})$
- ▶ The **filter tensor $\mathcal{H} = [\mathbf{H}_1, \dots, \mathbf{H}_L]$** is the trainable parameter.

- ▶ Illustrate definition with an FCNN with 3 layers

- ▶ Feed input signal $x = x_0$ into Layer 1

$$x_1 = \sigma[z_1] = \sigma[H_{1k} x_0]$$

- ▶ Output $\Phi(x; \mathcal{H})$ Parametrized by $\mathcal{H} = [H_1, H_2, H_3]$

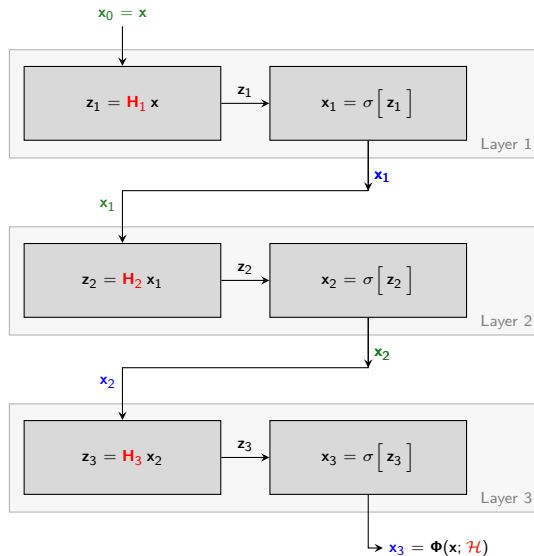


- ▶ Illustrate definition with an FCNN with 3 layers

- ▶ Feed Layer 1 output as an input to Layer 2

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

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

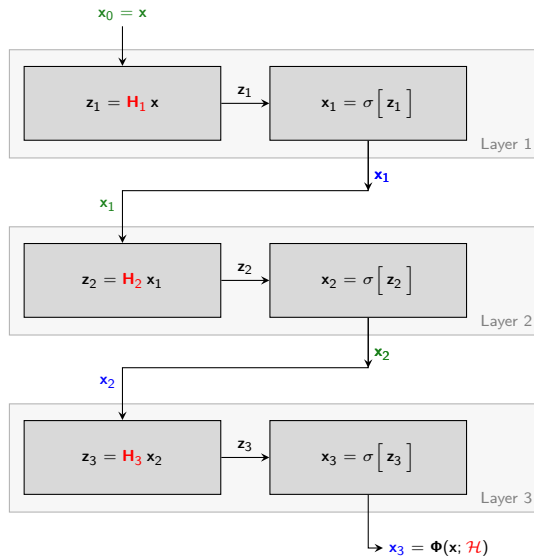


- ▶ Illustrate definition with an FCNN with 3 layers

- ▶ Feed Layer 2 output as an input to Layer 3

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

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



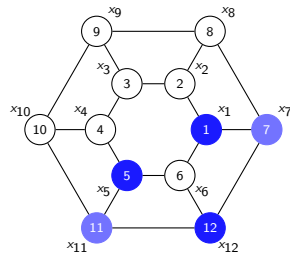
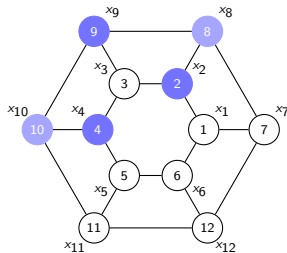
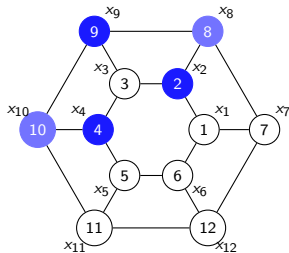
Neural Networks vs Graph Neural Networks

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

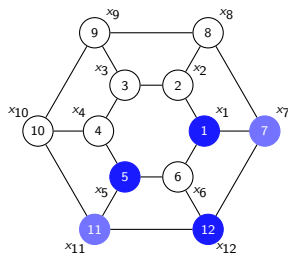
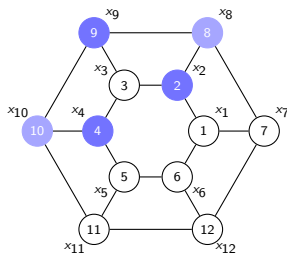
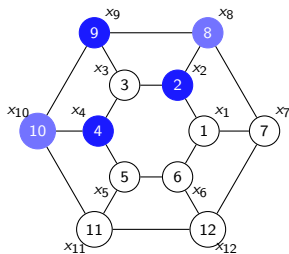
$$\min_{\mathcal{H}} \sum_{(x,y) \in \mathcal{T}} \ell(\Phi(x; \mathcal{H}), \mathbf{y}) \leq \min_{\mathcal{H}} \sum_{(x,y) \in \mathcal{T}} \ell(\Phi(x; \mathbf{S}, \mathcal{H}), \mathbf{y})$$

- ▶ 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
 - ⇒ Because it **exploits internal symmetries** of graph signals codified in the graph shift operator

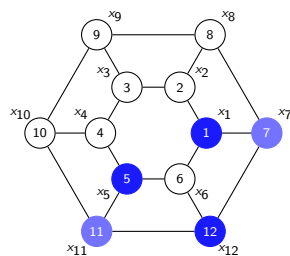
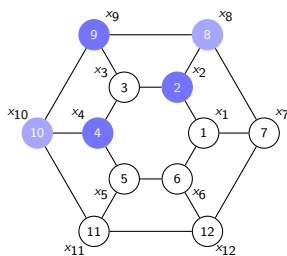
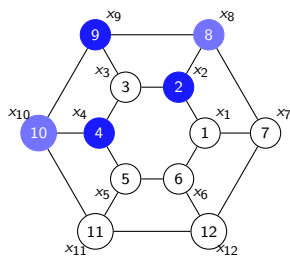
- ▶ Suppose the graph represents a recommendation system where we want to fill empty ratings
- ▶ We observe ratings with the structure in the left. But we do not observe examples like the other two
- ▶ From examples like the one in the left, the NN learns how to fill the middle signal but not the right



- ▶ 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



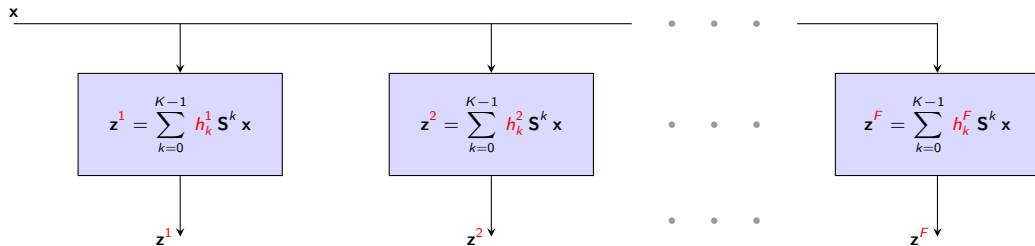
- ▶ The GNN exploits **symmetries** of the signal to effectively **multiply available data**
- ▶ This will be formalized later as the **permutation equivariance of graph neural networks**



Graph Filter Banks

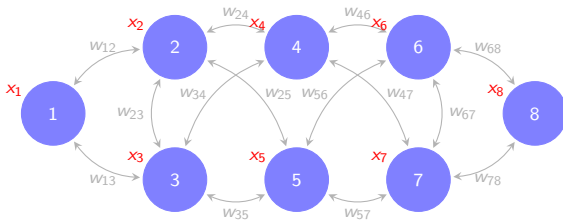
- ▶ Filters isolate features. When we are interested in multiple features, we use banks of filters

- ▶ 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** $\mathbf{h}^f = [h_1^f; \dots; h_{K-1}^f]$ \Rightarrow Output \mathbf{z}^f is a graph signal



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

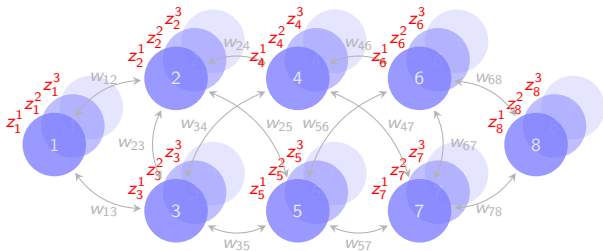
- ▶ The input of a filter bank is a single graph signal \mathbf{x} . Rows of \mathbf{x} are signals components x_i
- ▶ Output matrix \mathbf{Z} is a **collection of signals \mathbf{z}^f** . Rows of which are components \mathbf{z}_i^f
- ▶ Vector \mathbf{z}_i supported at each node. **Columns of \mathbf{Z}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



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

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 \mathbf{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**

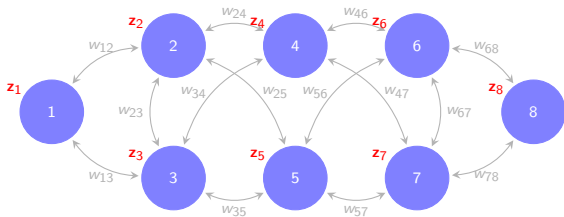


$$\mathbf{Z} = \begin{bmatrix} z_1^1 & \cdots & z_1^f & \cdots & z_1^F \\ \vdots & & \vdots & & \vdots \\ z_i^1 & \cdots & z_i^f & \cdots & z_i^F \\ \vdots & & \vdots & & \vdots \\ z_{N_V}^1 & \cdots & z_{N_V}^f & \cdots & z_{N_V}^F \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_i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

$$= [\mathbf{z}^1 \quad \cdots \quad \mathbf{z}^f \quad \cdots \quad \mathbf{z}^F]$$

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**



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

$$= [\mathbf{z}^1 \quad \cdots \quad \mathbf{z}^f \quad \cdots \quad \mathbf{z}^F]$$

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

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

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

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

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

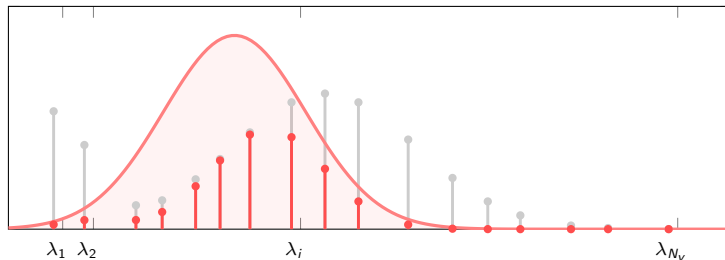
► We know that graph filters are **pointwise in the frequency domain** $\Rightarrow \tilde{z}_i = \tilde{h}(\lambda_i) \tilde{x}_i$

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

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

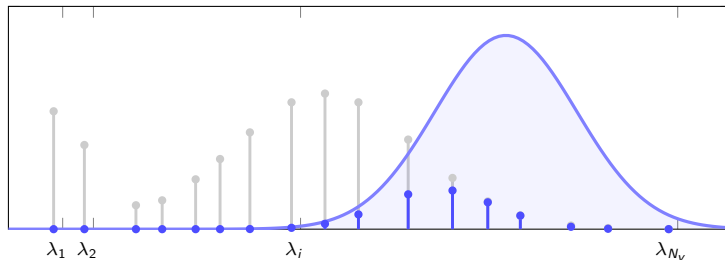
► But we have just seen the GFT preserves energy $\Rightarrow \|\mathbf{z}\|^2 = \|\tilde{\mathbf{z}}\|^2 = \sum_{i=1}^{N_v} \left(\tilde{h}(\lambda_i) \tilde{x}_i \right)^2$ ■

- ▶ 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



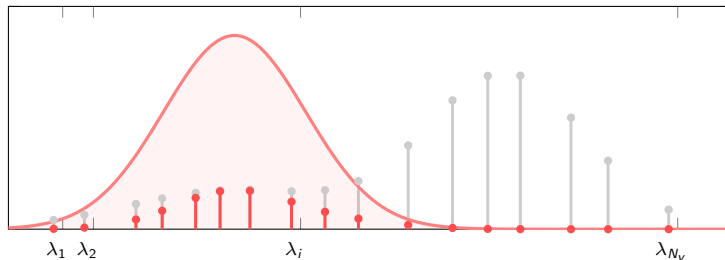
- ▶ Filter banks **scatter** the energy of signal \mathbf{x} into the signals \mathbf{z}^f at the output of the filters
⇒ Different signals concentrate energy on different outputs \mathbf{z}^f

- ▶ 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



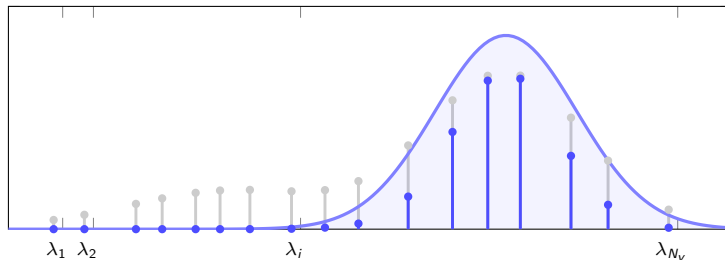
- ▶ Filter banks **scatter** the energy of signal \mathbf{x} into the signals \mathbf{z}^f at the output of the filters
⇒ Different signals concentrate energy on different outputs \mathbf{z}^f

- ▶ 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
⇒ Different signals concentrate energy on different outputs \mathbf{z}^f

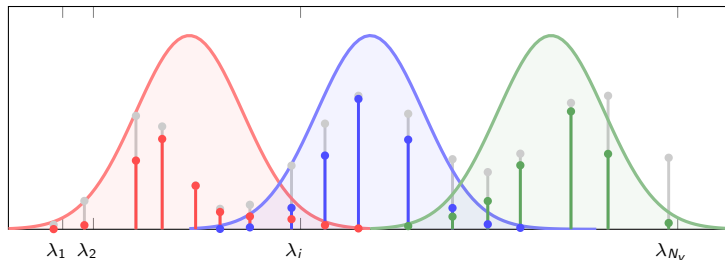
- ▶ The energy that graph filters let pass is a sort of “**area under the frequency response curve**”
- ▶ Graph Filter banks are helpful in identifying **frequency signatures** of different signals



- ▶ Filter banks **scatter** the energy of signal \mathbf{x} into the signals \mathbf{z}^f at the output of the filters
⇒ Different signals concentrate energy on different outputs \mathbf{z}^f

- ▶ The filter bank **isolates groups of frequency** components

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



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

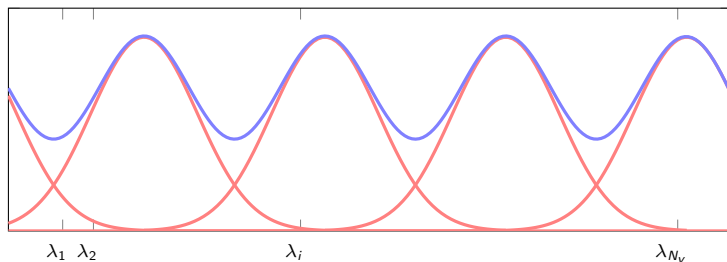
- ▶ The **GFT preserves energy** \Rightarrow It **scatters information**. But it **doesn't lose information**

- ▶ A filter bank is a **frame** if there exist constants $0 < m \leq M$ $\Rightarrow m \|\mathbf{x}\|^2 \leq \sum_{f=1}^F \|\mathbf{z}^f\|^2 \leq M \|\mathbf{x}\|^2$

- ▶ A filter bank is a **tight frame** if $m = M = 1$ $\Rightarrow \|\mathbf{x}\|^2 = \sum_{f=1}^F \|\mathbf{z}^f\|^2$

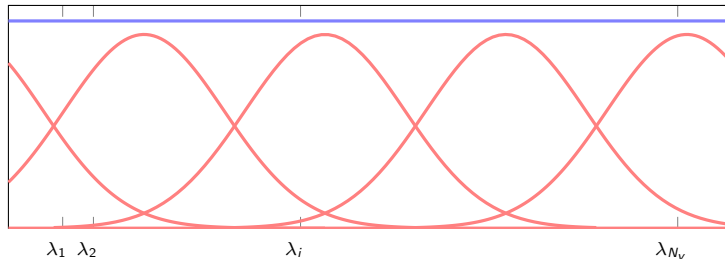
- ▶ No signal is vanquished by a frame. **Energy is preserved by a tight frame**

- ▶ Because filters are pointwise in the GFT domain, a frame must satisfy $\Rightarrow m \leq \sum_{f=1}^F [\tilde{h}^f(\lambda)]^2 \leq M$
- ▶ All frequencies λ must have at least one filter \mathbf{h}^f with response $m \leq [\tilde{h}^f(\lambda)]^2$



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

▶ A sufficient condition is that all frequencies accumulate unit energy when summing across all filters

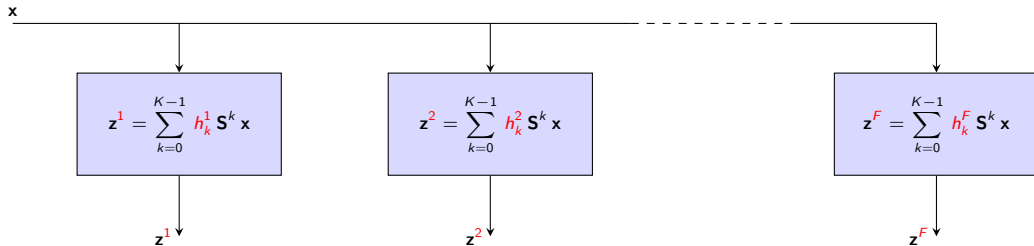


▶ We will not design filter banks. We will learn them. But keeping them close to frames is good

Multiple Feature GNNs

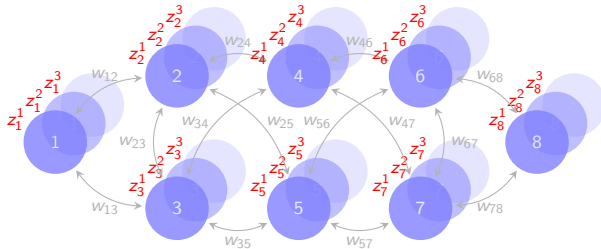
- ▶ We leverage filter banks to create GNNs that process multiple features per layer

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



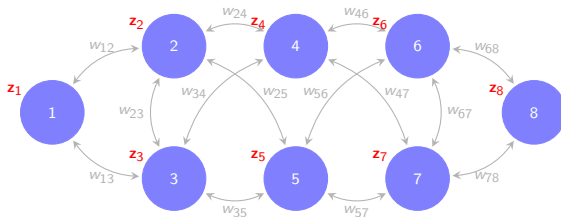
- ▶ We would now like to **process** multiple feature graph signals. Process each feature with a filterbank

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



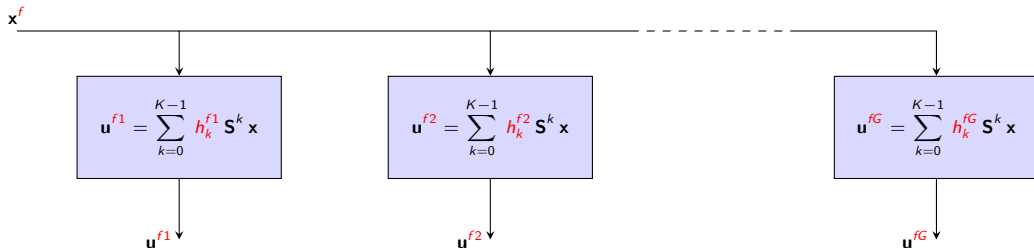
- ▶ We would now like to **process** multiple feature graph signals. Process each feature with a filterbank

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

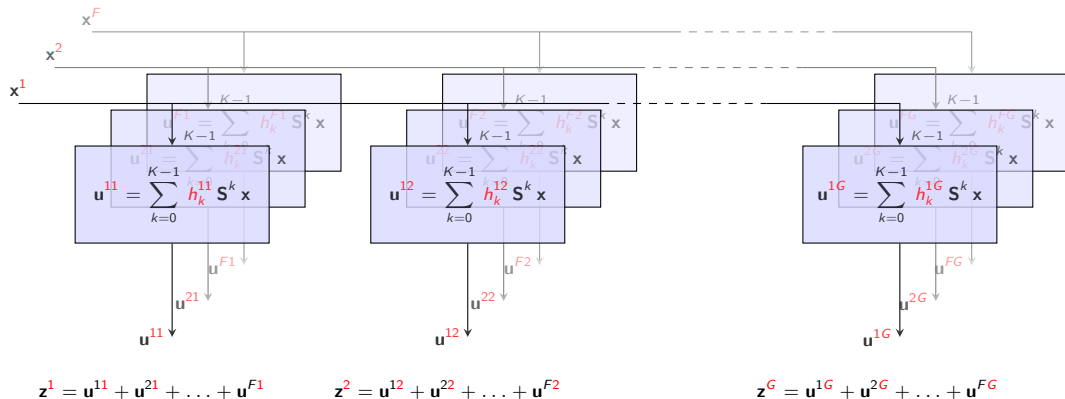


- ▶ We would now like to **process** multiple feature graph signals. Process each feature with a filterbank

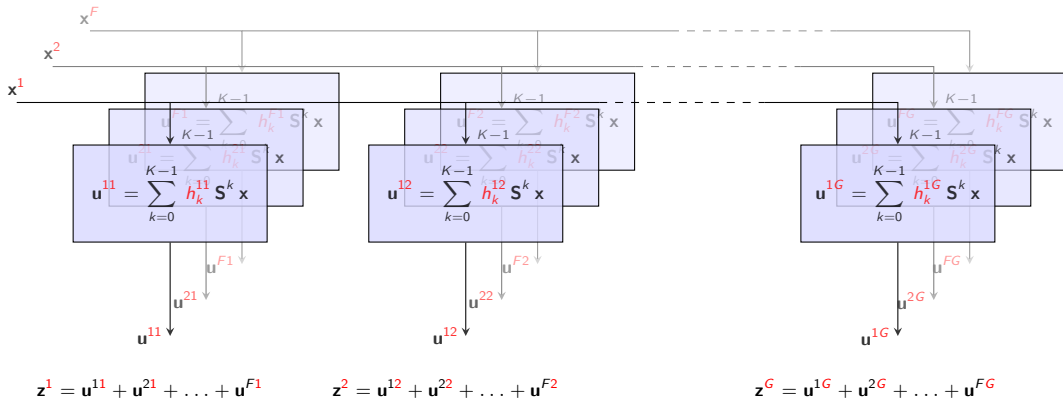
- Each of the F features \mathbf{x}^f is processed with G filters with coefficients $h_k^{fg} \Rightarrow \mathbf{u}^{fg} = \sum_{k=0}^{K-1} h_k^{fg} \mathbf{S}^k \mathbf{x}^f$



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



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



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

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

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

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

- ▶ MIMO GNN stacks MIMO perceptrons \Rightarrow Compose of MIMO filters with pointwise nonlinearities
- ▶ Layer ℓ processes input signal $\mathbf{X}_{\ell-1}$ with perceptron $\mathbf{H}_\ell = [\mathbf{H}_{\ell 0}, \dots, \mathbf{H}_{\ell, K-1}]$ to produce output \mathbf{X}_ℓ

$$\mathbf{X}_\ell = \sigma[\mathbf{z}_\ell] = \sigma \left[\sum_{k=0}^{K-1} \mathbf{s}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k} \right]$$

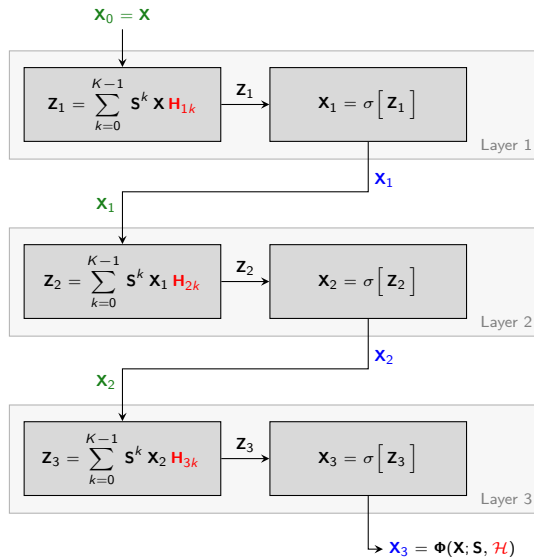
- ▶ Denoting the Layer 1 input as $\mathbf{X}_0 = \mathbf{X}$, this provides a recursive definition of a MIMO GNN
- ▶ If it has L layers, the GNN output $\Rightarrow \mathbf{X}_L = \Phi(\mathbf{x}; \mathbf{S}, \mathbf{H}_1, \dots, \mathbf{H}_L) = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$
- ▶ The filter tensor $\mathcal{H} = [\mathbf{H}_1, \dots, \mathbf{H}_L]$ 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)

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

- ▶ Last layer output is the GNN output $\Rightarrow \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H})$
- \Rightarrow Parametrized by trainable tensor $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$

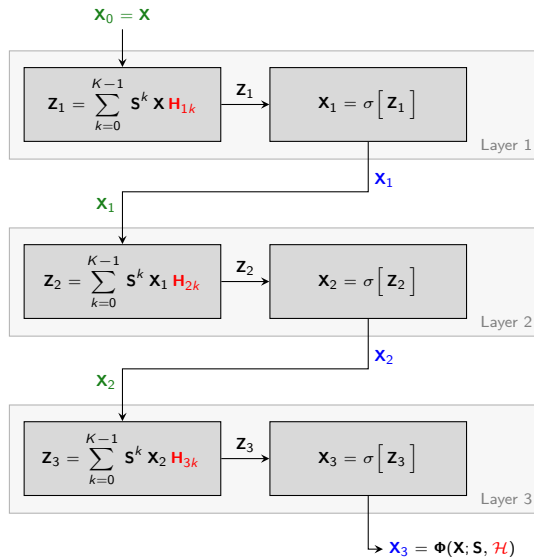


MIMO GNN Block Diagram

- ▶ We illustrate with a MIMO GNN with 3 layers
- ▶ Feed **Layer 1** output as an **input** to **Layer 2** (F_1 features)

$$\mathbf{x}_2 = \sigma[\mathbf{z}_2] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{s}^k \mathbf{x}_1 \mathbf{H}_{2k}\right]$$

- ▶ 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

$$\mathbf{x}_3 = \sigma[\mathbf{z}_3] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{s}^k \mathbf{x}_2 \mathbf{H}_{3k}\right]$$

- ▶ Last layer output is the GNN output $\Rightarrow \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H})$
- \Rightarrow Parametrized by trainable tensor $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$

