max planck institut informatik

SIC Saarland Informatics Campus

Probabilistic Graphical Models and Their Applications

Graph Neural Networks - Lecture 2

@ Jan 20, 2021

Bernt Schiele

www.mpi-inf.mpg.de/gm/

Max Planck Institute for Informatics & Saarland University,
Saarland Informatics Campus Saarbrücken

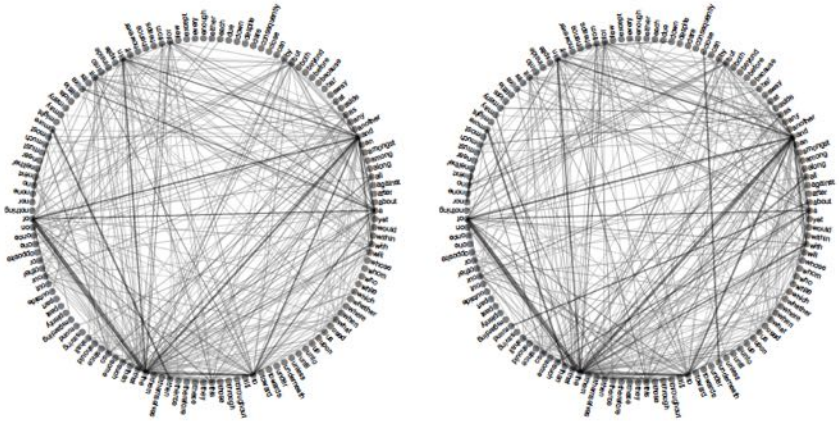
Overview Today's Lecture

- Brief Recap
- Graph Signals & Graph Convolutional Filters
- Graph Neural Networks vs Fully Connected Graph Networks
- Permutation Equivariance

Graphs are Common...

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

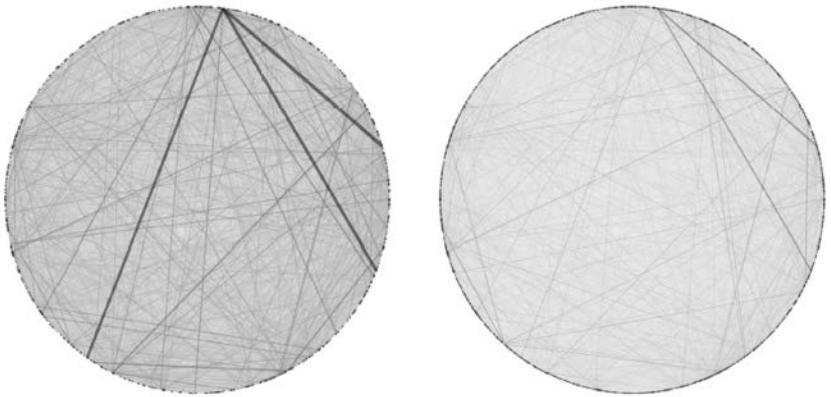
Authorship Attribution



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

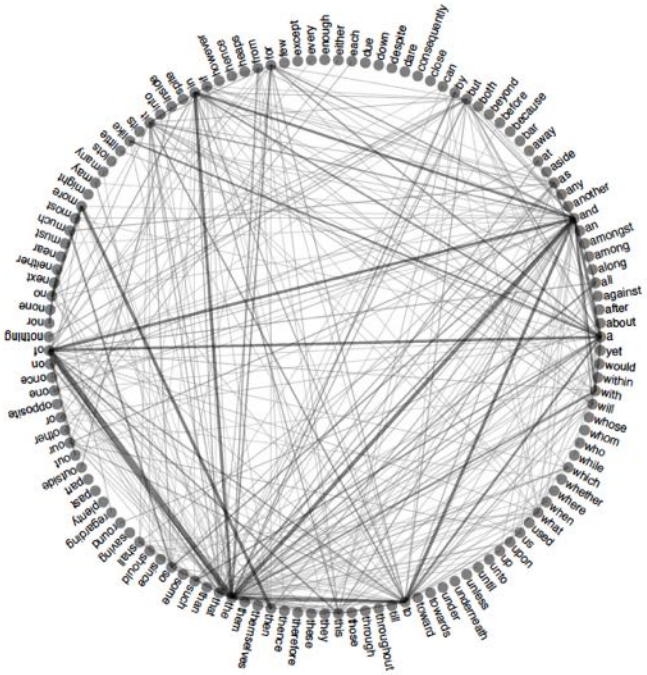
slide credit: Alejandro Ribeiro

Authorship Attribution with Word Adjacency Networks

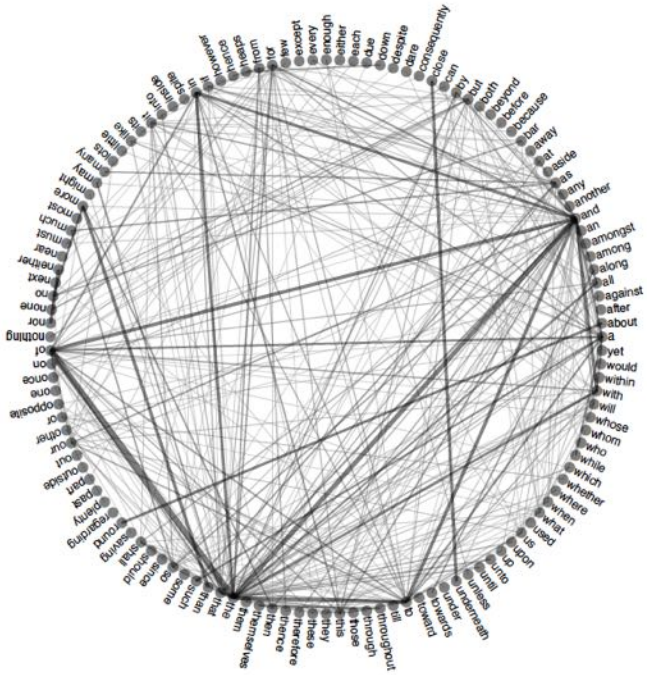
- Nodes represent different function words and edges how often words appear close to each other

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

William Shakespeare



Christopher Marlowe



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

slide credit: Alejandro Ribeiro

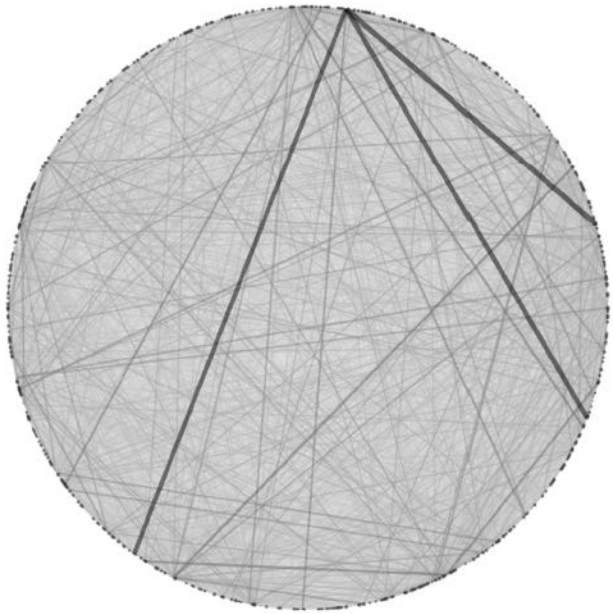
Segarra-Eisen-Egan-Ribeiro, *Attributing the Authorship of the Henry VI Plays by Word Adjacency*, Shakespeare Quarterly 2016, doi.org/10.1353/shq.2016.0024

Recommendation System with Collaborative Filtering

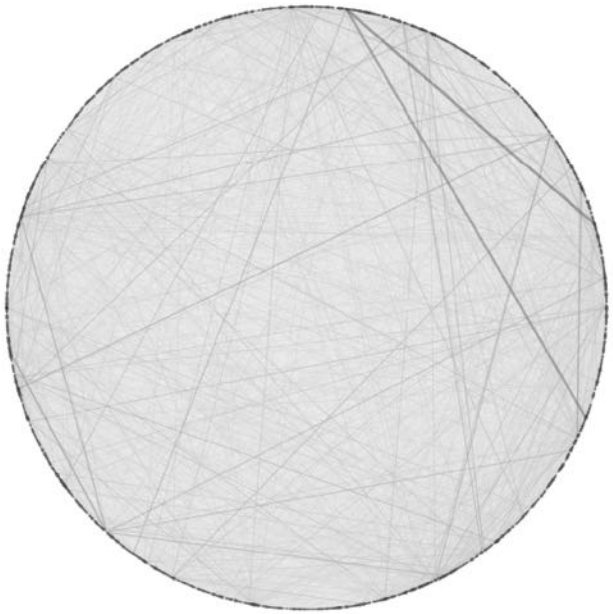
- Nodes represent different customers and edges their average similarity in product ratings

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

Variation Diagram for Original (sampled) ratings



Variation Diagram for Reconstructed (predicted) ratings



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

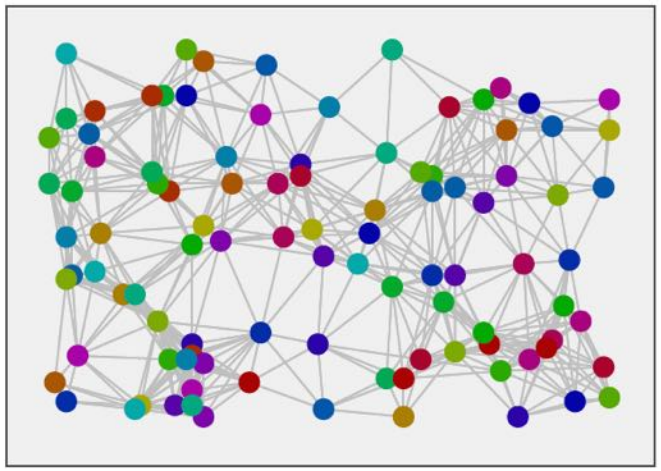
slide credit: Alejandro Ribeiro

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

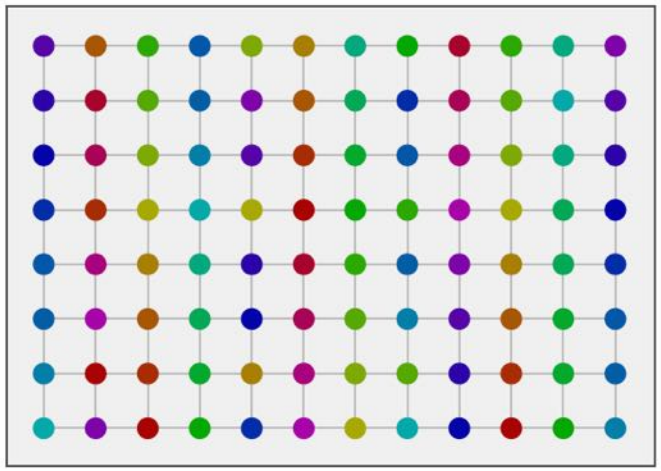
Neural Networks and Convolutional Neural Networks

- 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



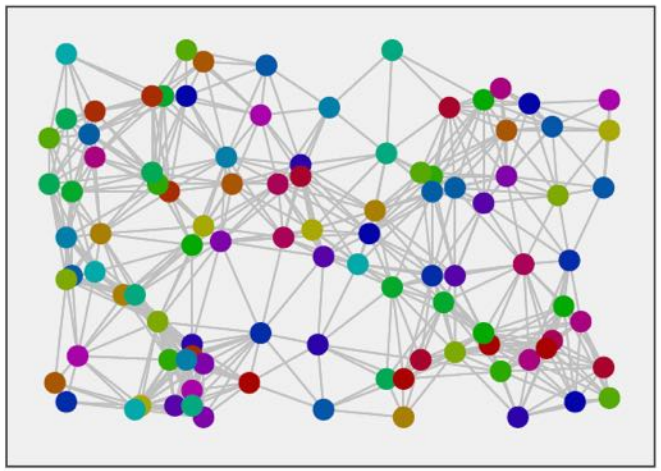
- Generic NNs do not scale to large dimensions $\Rightarrow$ Convolutional Neural Networks (CNNs) do scale

slide credit: Alejandro Ribeiro

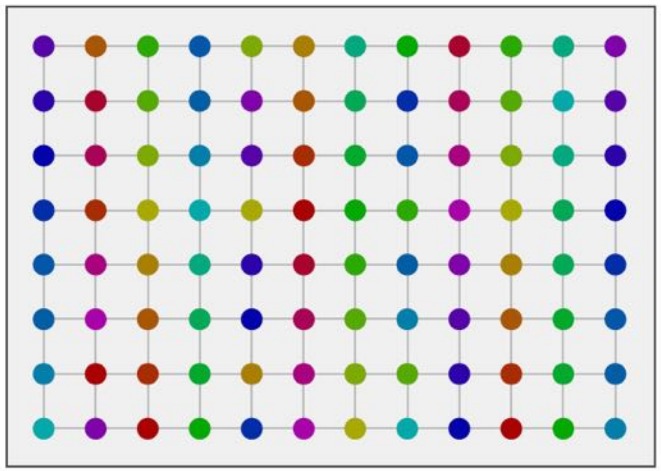
Convolutional Neural Networks and Graph Neural Networks

- 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)

slide credit: Alejandro Ribeiro

Convolutional Neural Networks and Graph Neural Networks

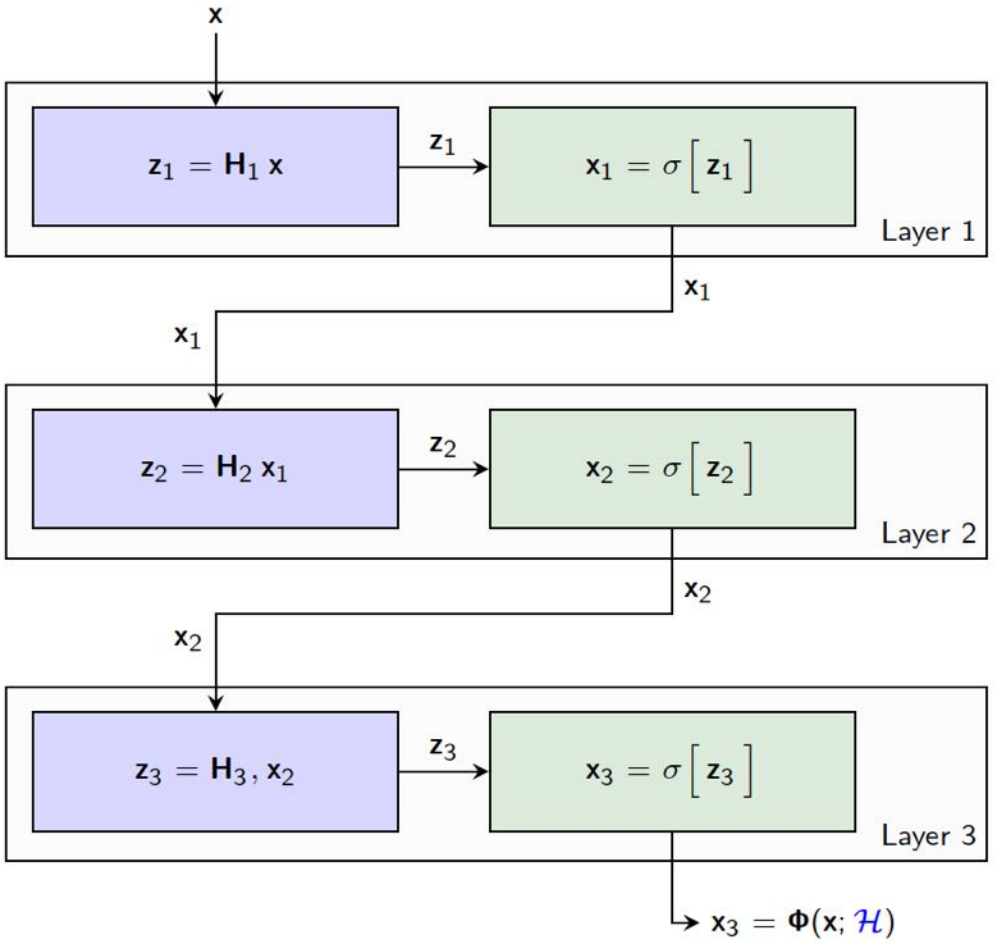
- CNNs and GNNe are minor variations of linear convolutional filters

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

slide credit: Alejandro Ribeiro

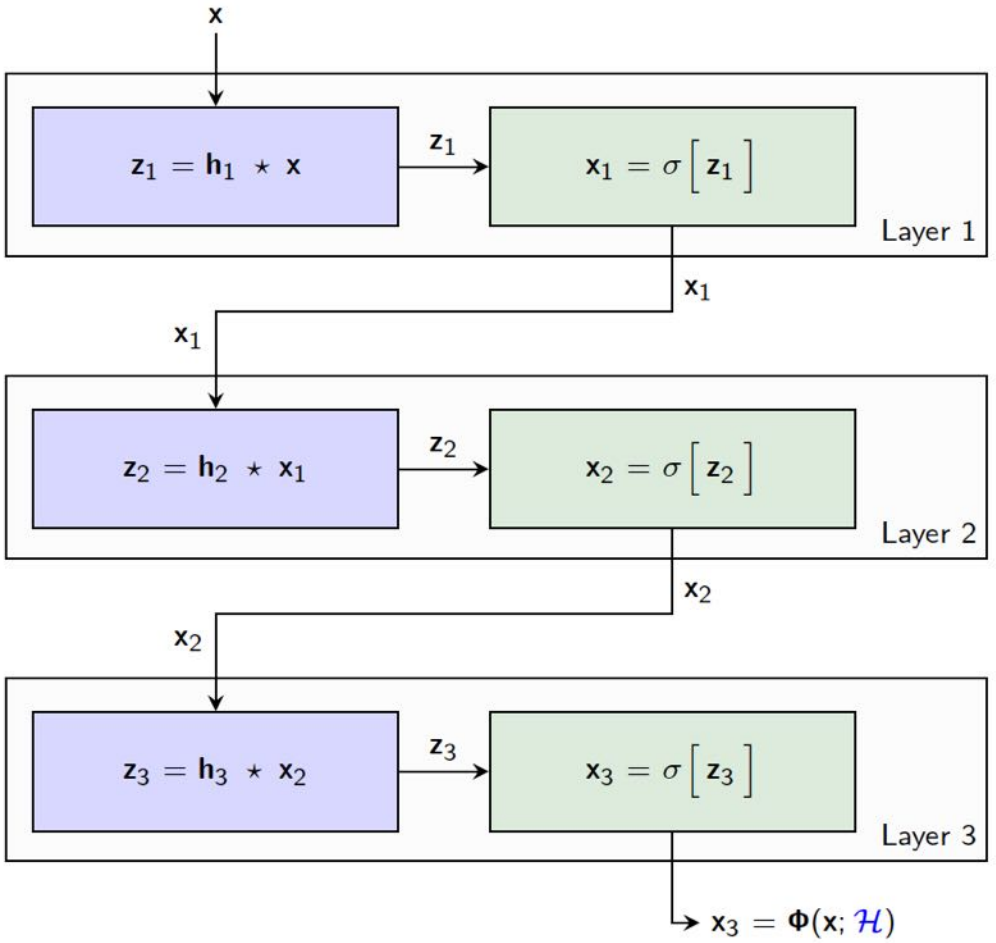
Neural Networks

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



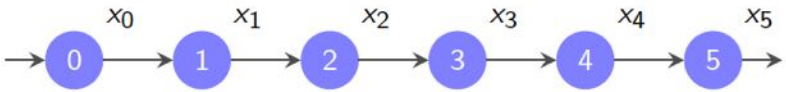
Convolutional Neural Networks (CNNs)

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

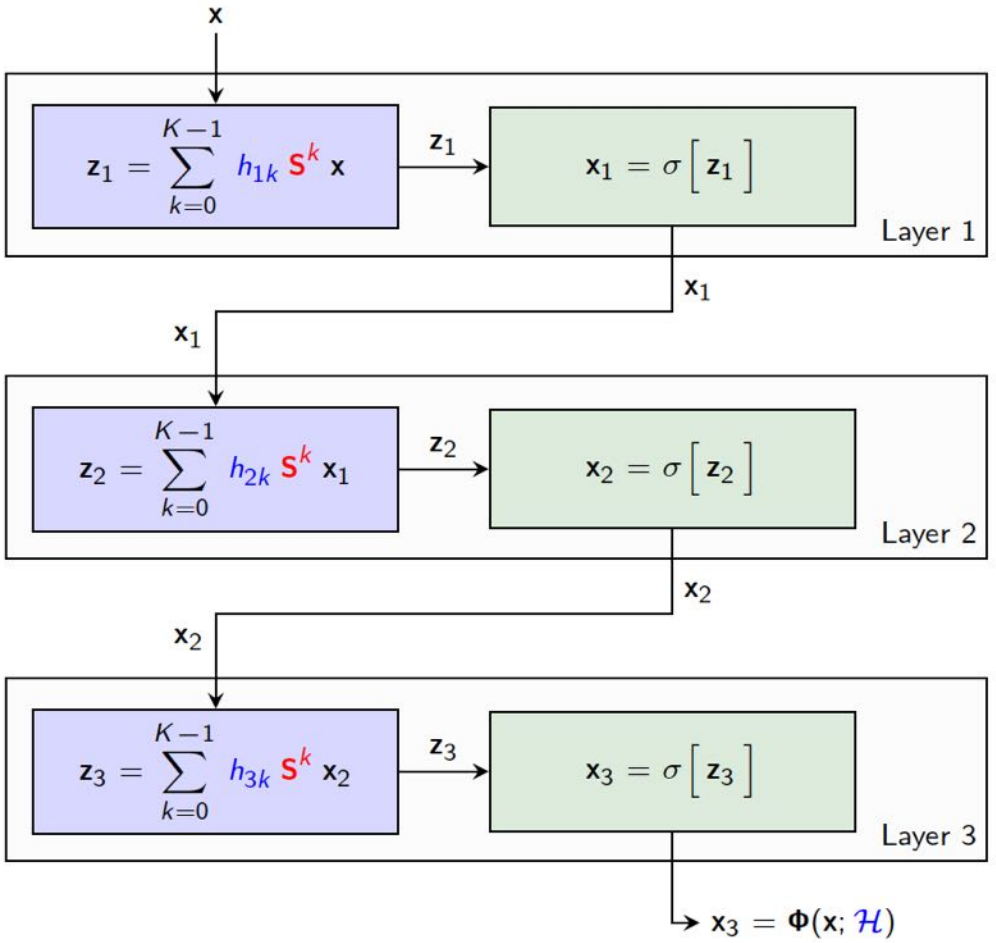


When we Think of Time Signal as Supported by a Line Graph

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

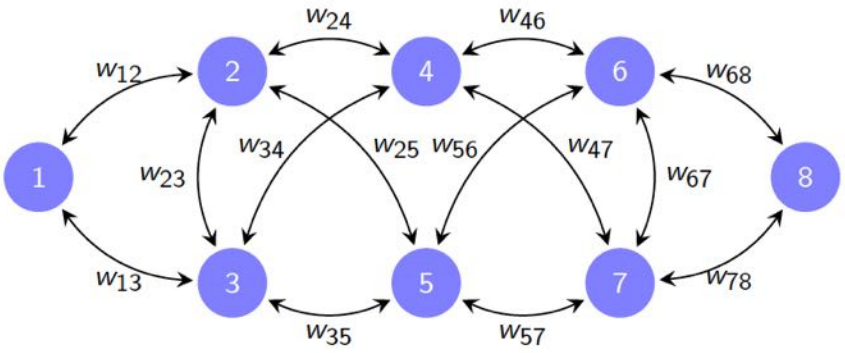


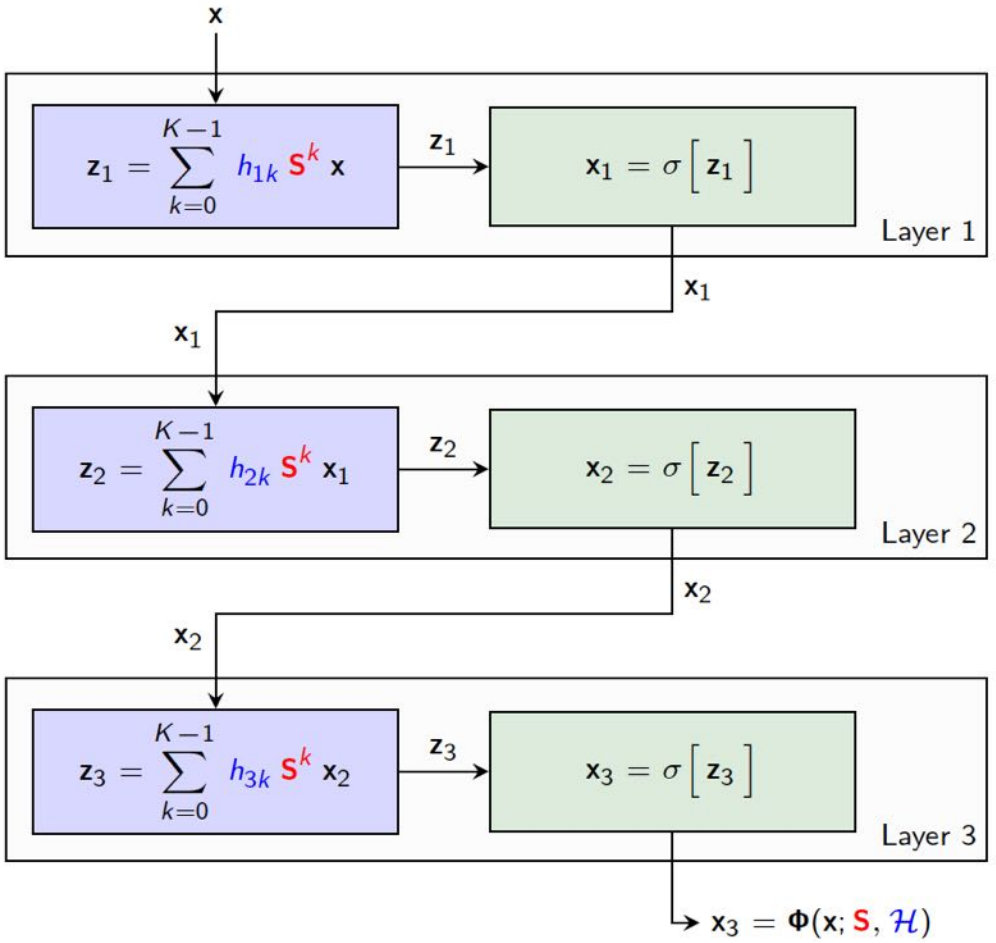
- Just another way of writing convolutions and Just another way of writing CNNs
- But one that lends itself to generalization



Graph Neural Networks (GNNs)

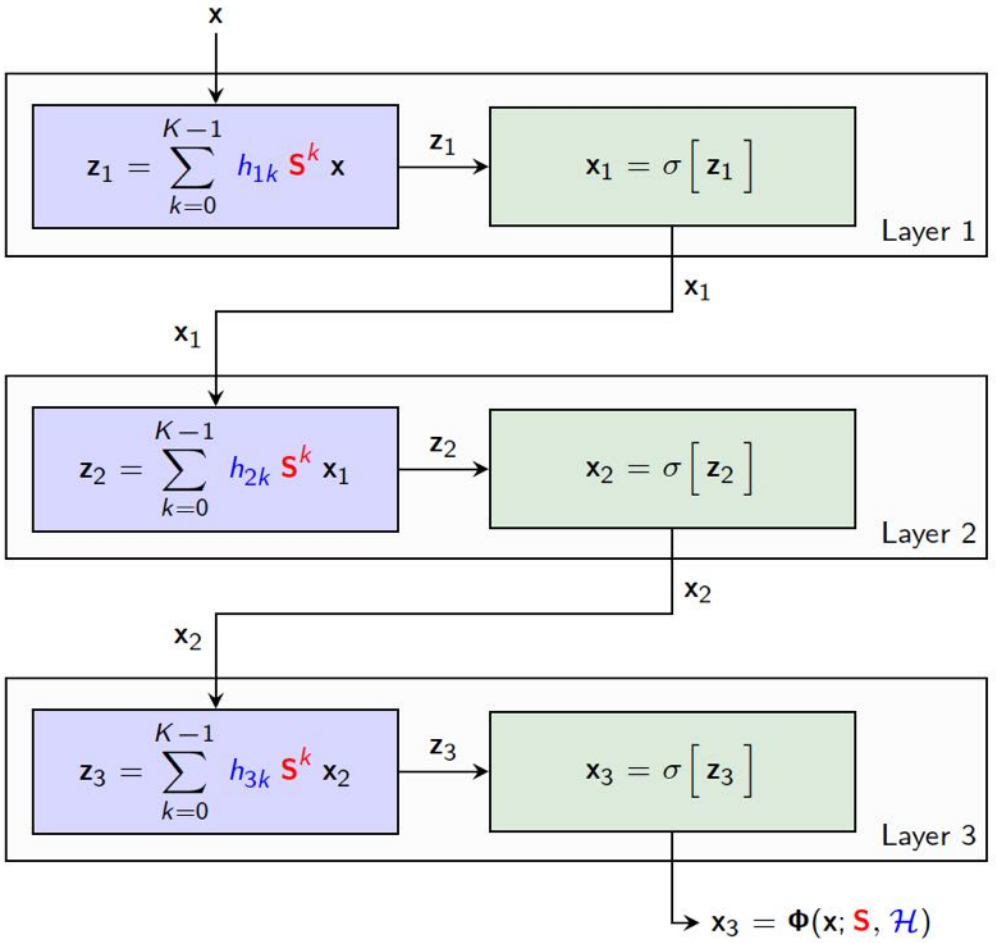
- The graph can be any arbitrary graph
- The polynomial on the matrix representation **S** becomes a graph convolutional filter





Graph Neural Networks (GNNs)

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



Graphs

slide credit: Alejandro Ribeiro

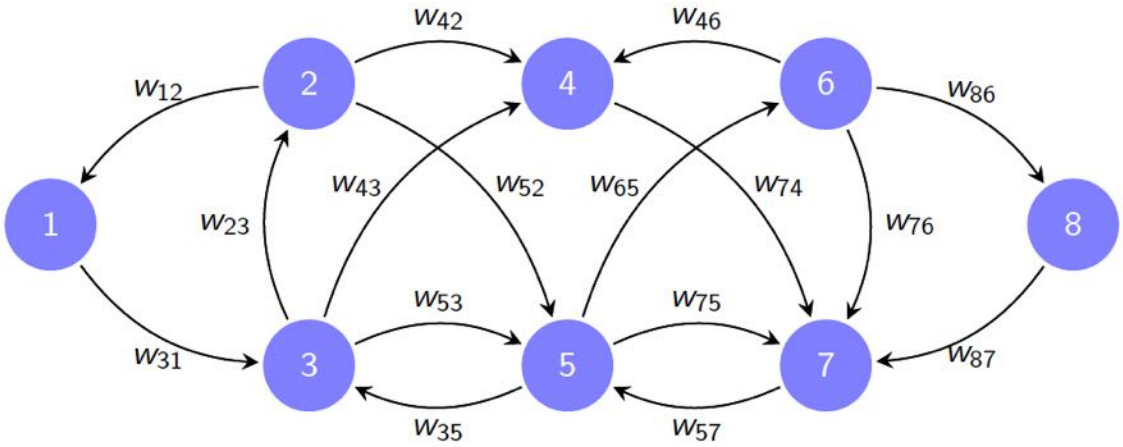
Nodes, Edges, Weights

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

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

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

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



slide credit: Alejandro Ribeiro

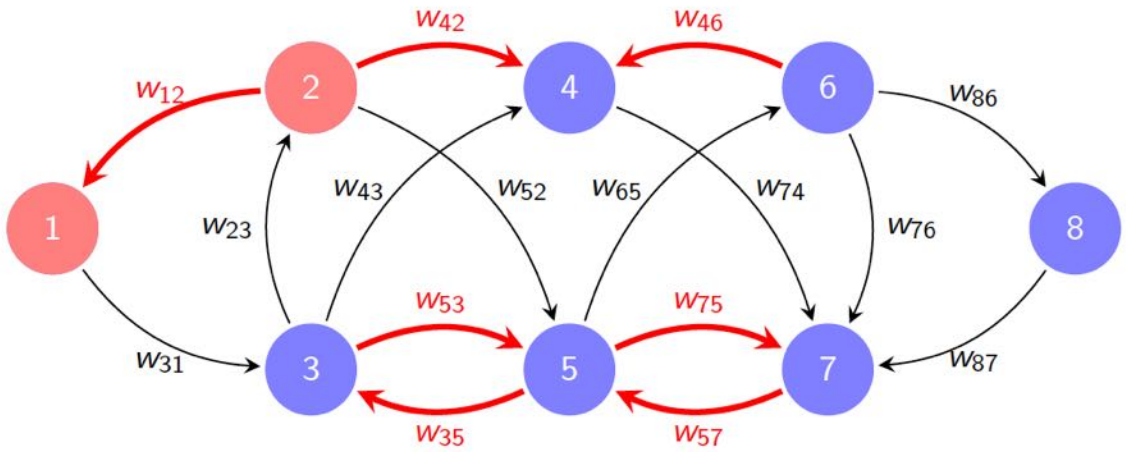
Directed Graphs

- 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 $w_{ij} \neq w_{ji}$



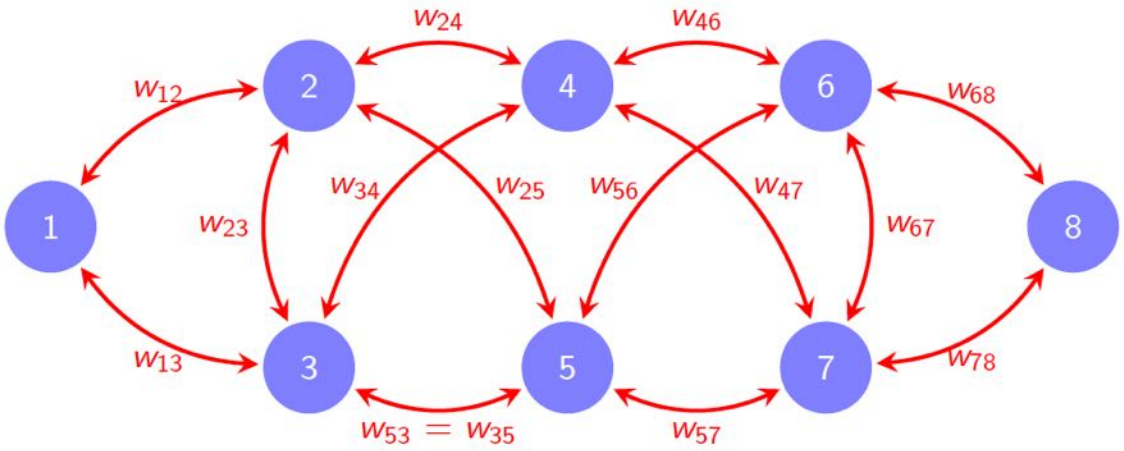
slide credit: Alejandro Ribeiro

Symmetric Graphs

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



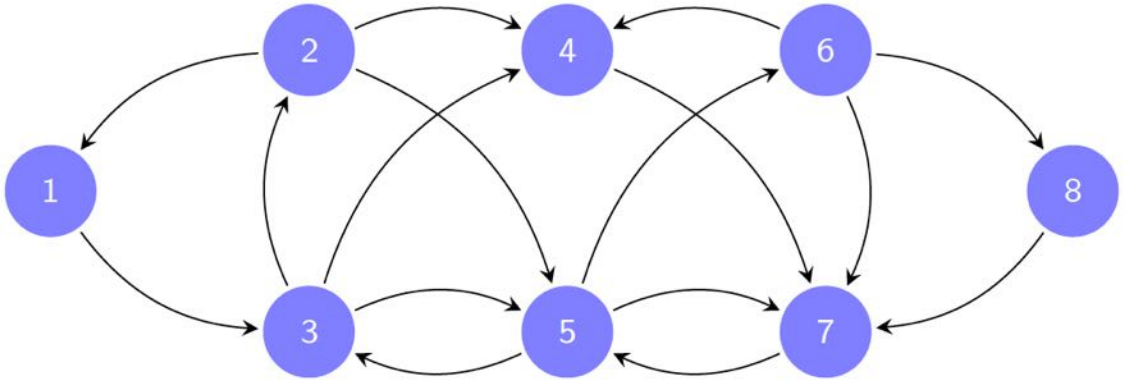
slide credit: Alejandro Ribeiro

Unweighted Graph

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

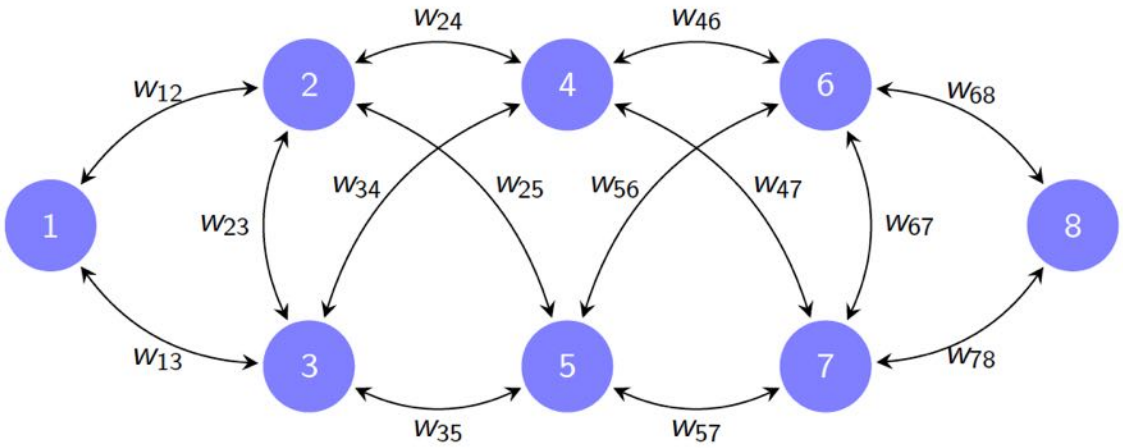
- Unweighted graphs could be directed or symmetric



slide credit: Alejandro Ribeiro

Weighted Symmetric Graph

- 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



slide credit: Alejandro Ribeiro

Graph Shift Operators

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

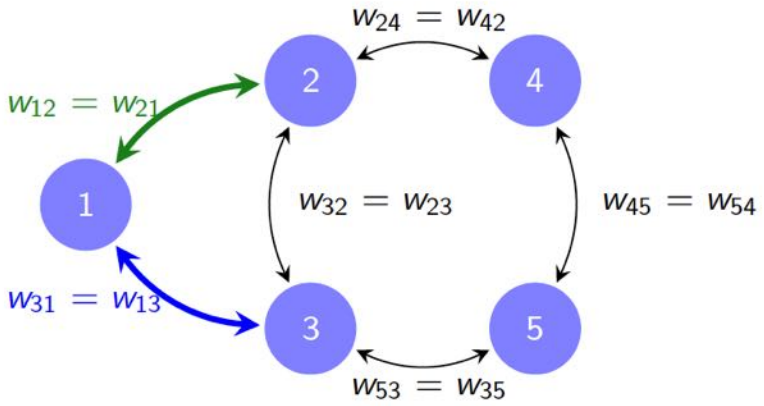
slide credit: Alejandro Ribeiro

Adjacency Matrix

- 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



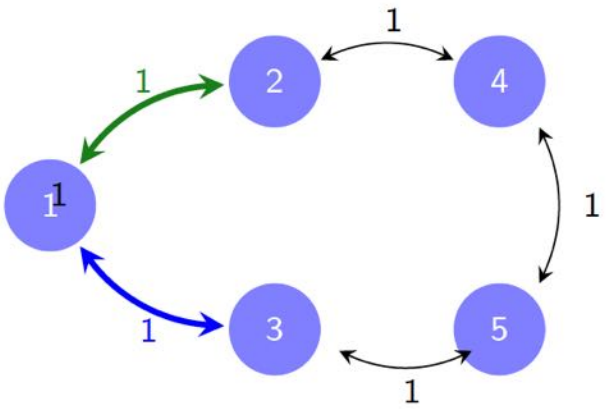
$$\mathbf{A} = \begin{bmatrix} 0 & w_{12} & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{23} & w_{24} & 0 \\ w_{31} & w_{32} & 0 & 0 & w_{35} \\ 0 & w_{42} & 0 & 0 & w_{45} \\ 0 & 0 & w_{53} & w_{54} & 0 \end{bmatrix}.$$

slide credit: Alejandro Ribeiro

Adjacency Matrix for Unweighted Graph

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

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

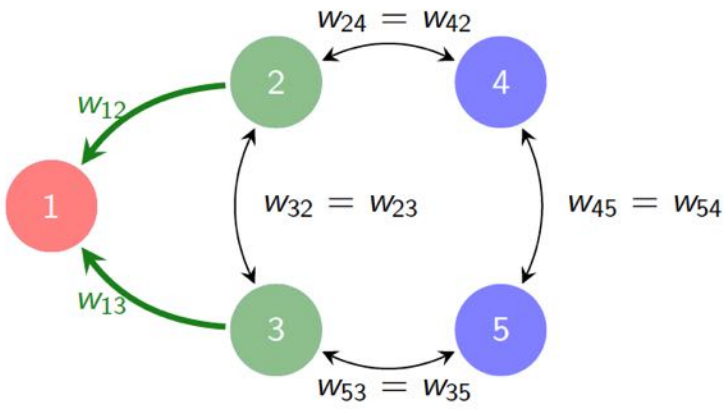


$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

slide credit: Alejandro Ribeiro

Neighborhood and Degree

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

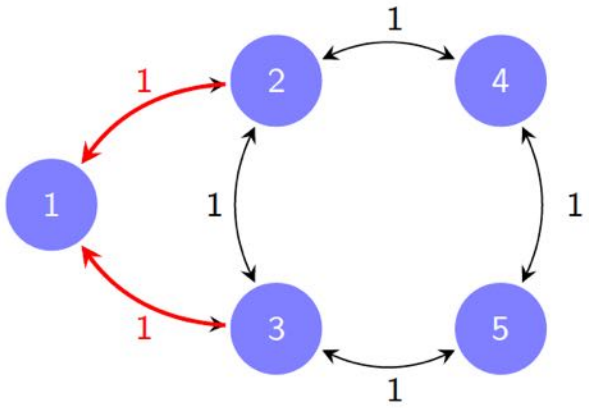


- Node 1 neighborhood $\Rightarrow n(1) = \{2, 3\}$
- Node 1 degree $\Rightarrow n(1) = w_{12} + w_{13}$

slide credit: Alejandro Ribeiro

Degree Matrix

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



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

slide credit: Alejandro Ribeiro

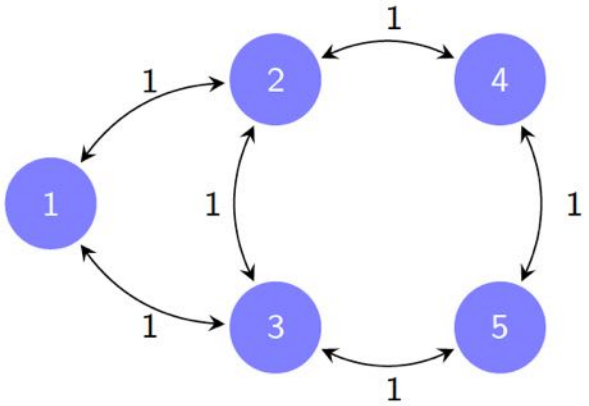
Laplacian Matrix

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

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



slide credit: Alejandro Ribeiro

Graph Shift Operator

- 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 $\mathbf{S}$

slide credit: Alejandro Ribeiro

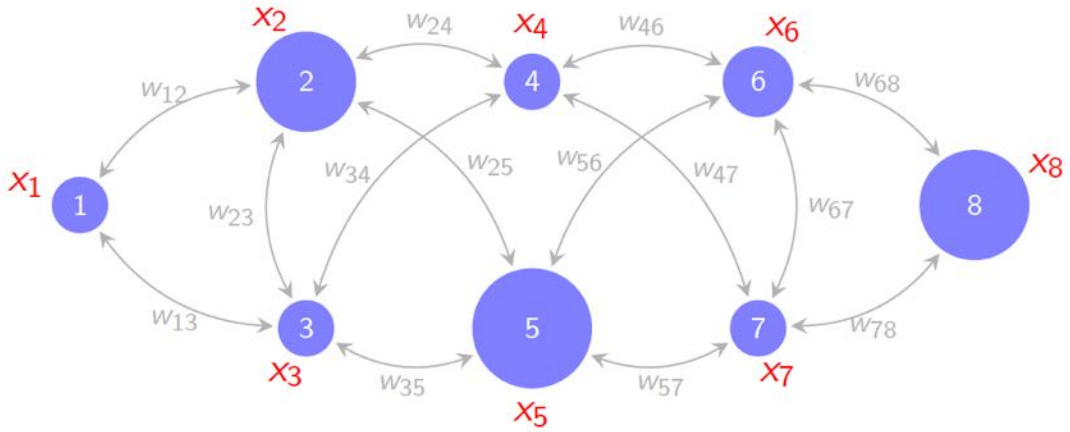
Graph Signals

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

slide credit: Alejandro Ribeiro

Graph Signals

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



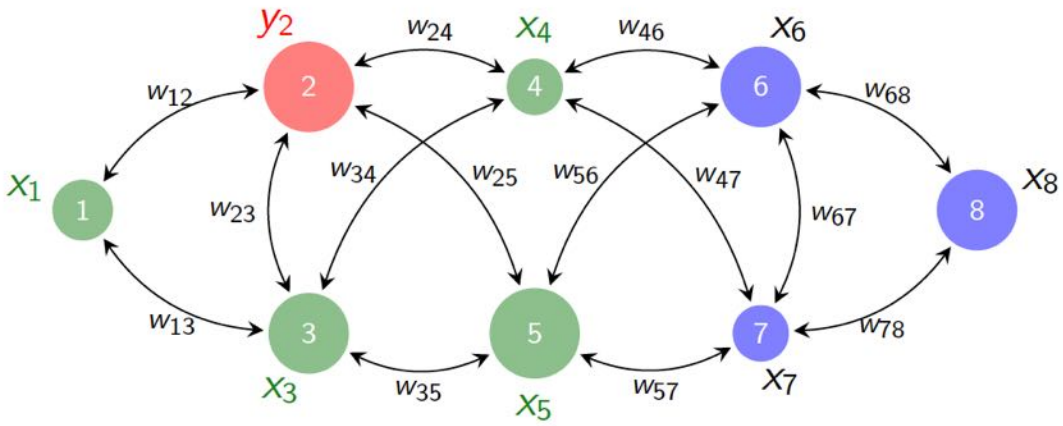
- The graph is an expectation of proximity or similarity between components of the signal $\mathbf{x}$

Graph Signal Diffusion

- 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.



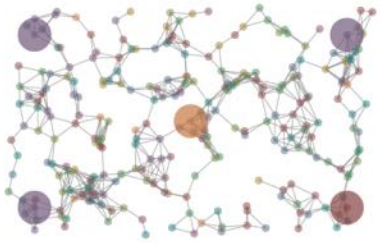
slide credit: Alejandro Ribeiro

Diffusion Sequence

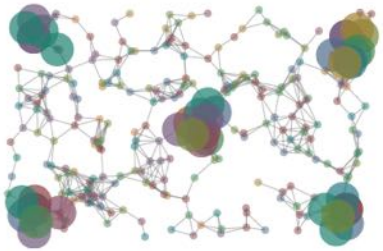
- Compose the diffusion operator to produce diffusion sequence $\Rightarrow$ defined recursively as

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

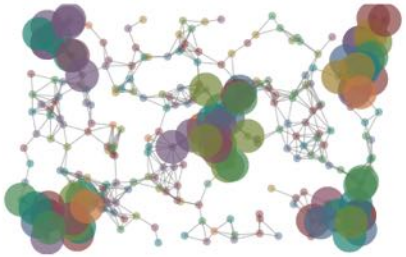
- Can unroll the recursion and write the diffusion sequence as the power sequence $\Rightarrow \mathbf{x}^{(k)} = \mathbf{S}^k\mathbf{x}$



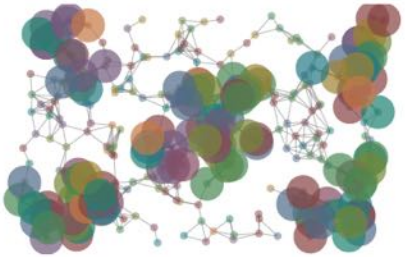
$\mathbf{x}^{(0)} = \mathbf{x} = \mathbf{S}^0\mathbf{x}$



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



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



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

slide credit: Alejandro Ribeiro

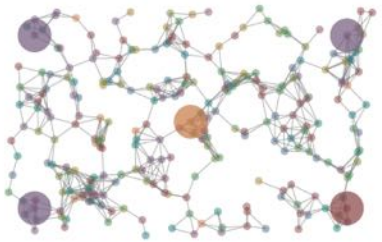
Observations about Diffusion Sequences

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

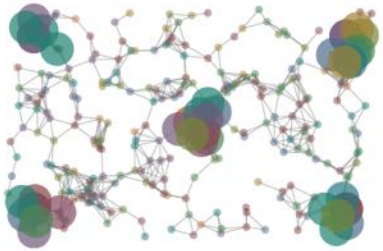
  $\Rightarrow$ One reason why we use the diffusion sequence to define graph convolutions

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

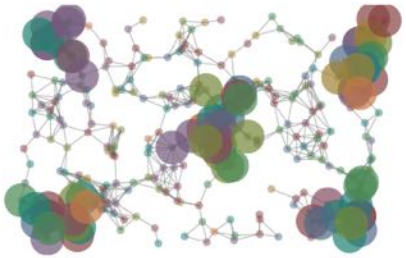
  $\Rightarrow$ Always implement the recursive version. The power version is good for analysis



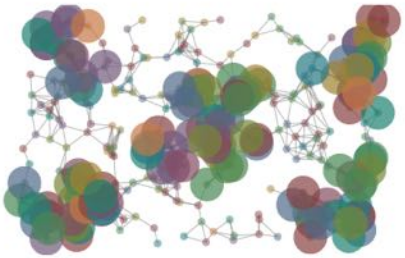
$\mathbf{x}^{(0)} = \mathbf{x} = \mathbf{S}^0\mathbf{x}$



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



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



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

slide credit: Alejandro Ribeiro

Graph Convolutional Filters

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

slide credit: Alejandro Ribeiro

Graph Filters

- 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

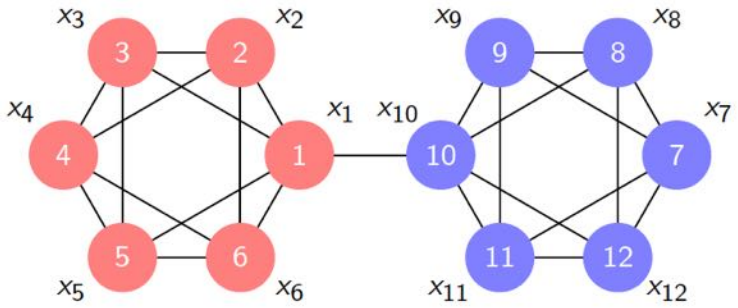
$$\mathbf{y} = \mathbf{H}(\mathbf{S})\,\mathbf{x} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$$

- 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 $\mathbf{x}$

slide credit: Alejandro Ribeiro

From Local to Global Information

- 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 $\mathbf{h} = \{h_k\}_{k=0}^{K-1}$



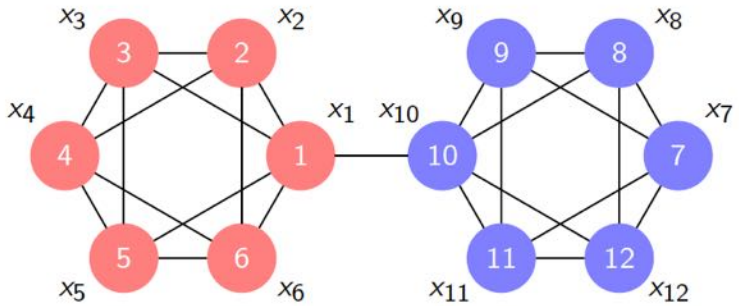
- Graph convolution output $\Rightarrow \mathbf{y} = \mathbf{h} \star_{\mathbf{S}} \mathbf{x} = h_0\mathbf{S}^0\mathbf{x} + h_1\mathbf{S}^1\mathbf{x} + h_2\mathbf{S}^2\mathbf{x} + h_3\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{K-1} h_k\mathbf{S}^k\mathbf{x}$

slide credit: Alejandro Ribeiro

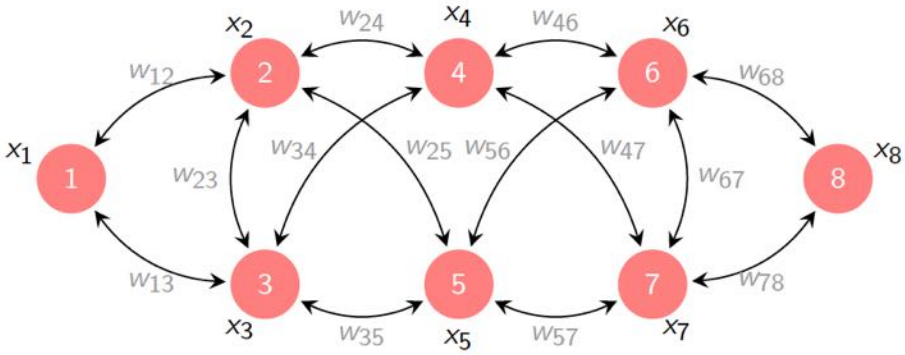
Transferability of Filters Across Graphs

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

Graph Filter on a Graph



Same Graph Filter on Another Graph



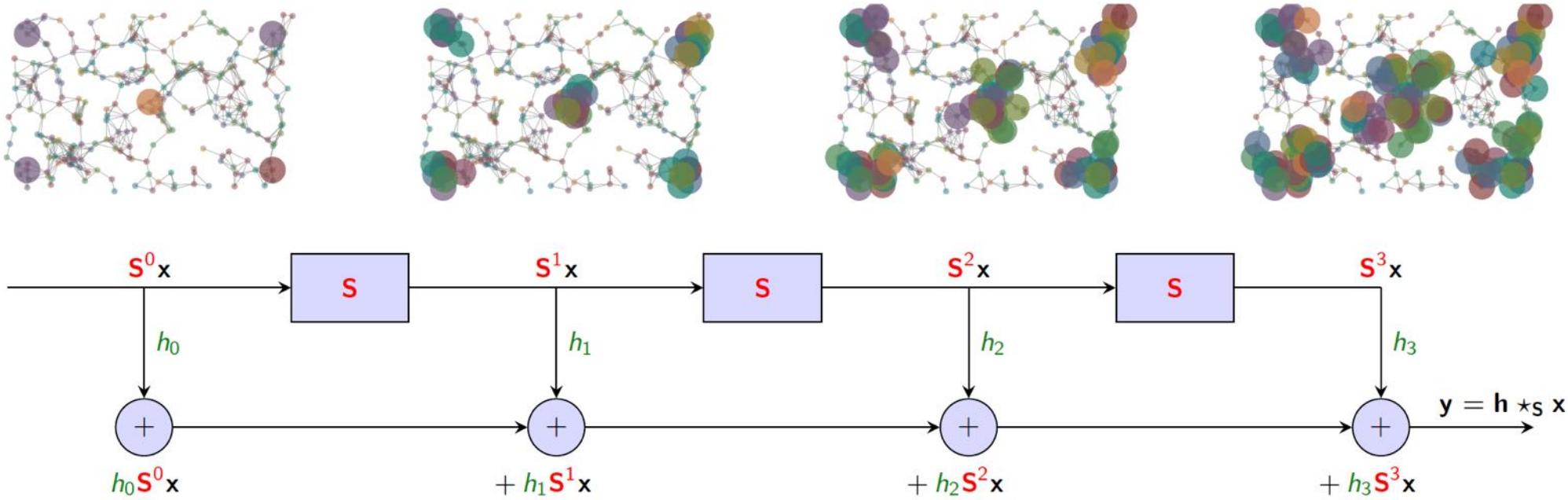
- Graph convolution output $\Rightarrow \mathbf{y} = \mathbf{h} \star_{\mathbf{S}} \mathbf{x} = h_0\mathbf{S}^0\mathbf{x} + h_1\mathbf{S}^1\mathbf{x} + h_2\mathbf{S}^2\mathbf{x} + h_3\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\mathbf{S}^k\mathbf{x}$

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

slide credit: Alejandro Ribeiro

Graph Convolution Filters as Diffusion Operators

- 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



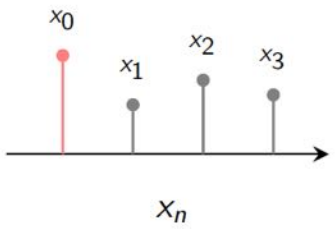
slide credit: Alejandro Ribeiro

Time Convolutions as a Particular Case of Graph Convolutions

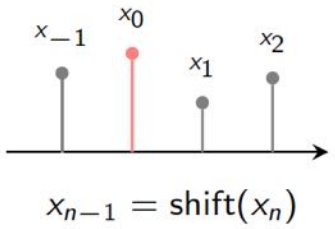
slide credit: Alejandro Ribeiro

Convolutions in Time

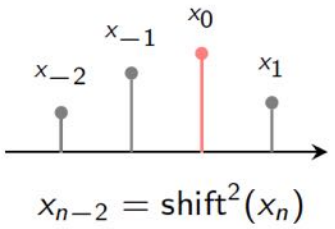
- Convolutional filters process signals in time by leveraging the time shift operator



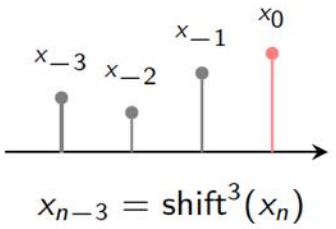
x_n



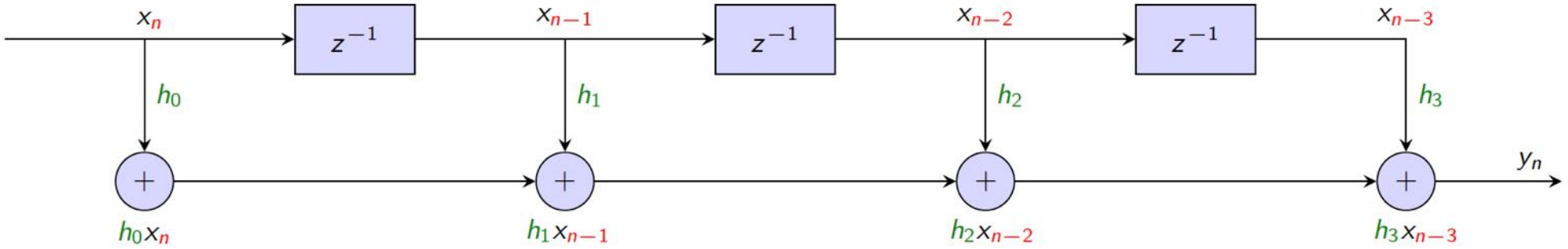
$x_{n-1} = \text{shift}(x_n)$



$x_{n-2} = \text{shift}^2(x_n)$



$x_{n-3} = \text{shift}^3(x_n)$

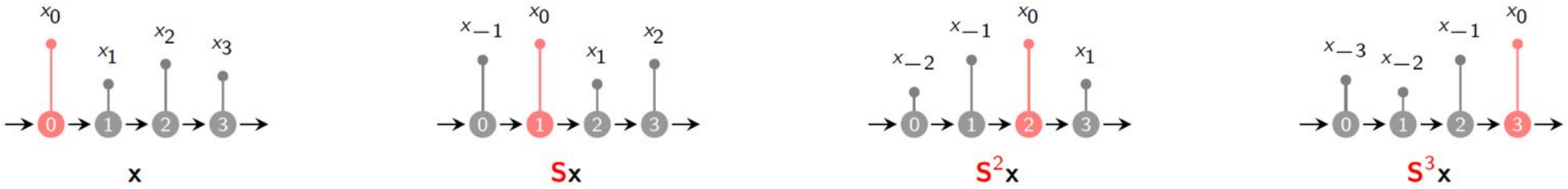


- The time convolution is a linear combination of time shifted inputs $\Rightarrow y_n = \sum_{k=0}^{K-1} h_k x_{n-k}$

slide credit: Alejandro Ribeiro

Time Signals Represented as Graph Signals

- 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

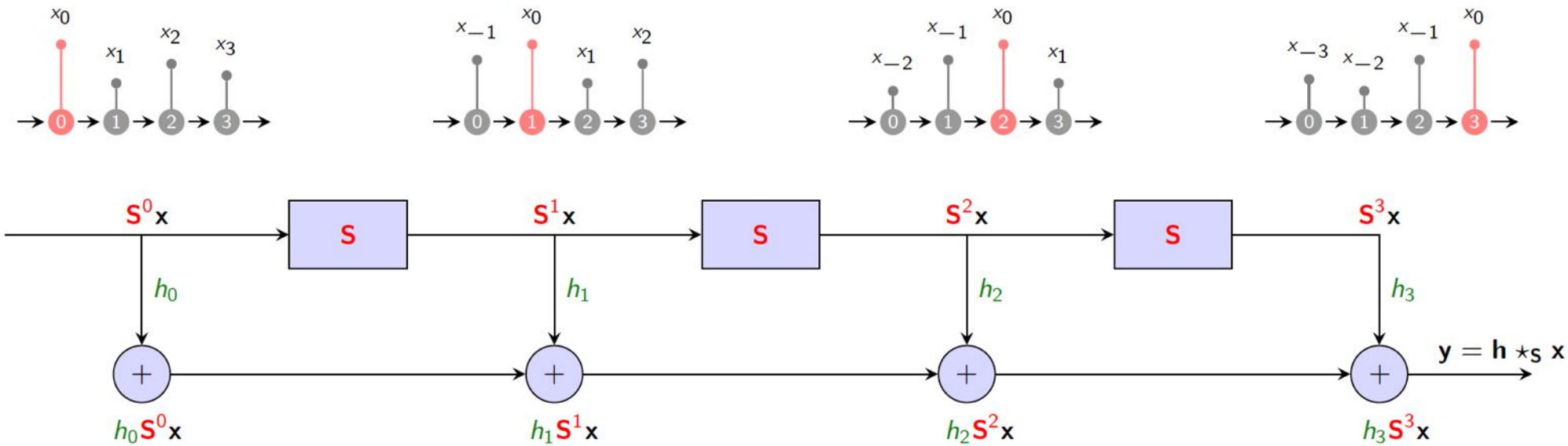
$$\mathbf{S}^3\mathbf{x} = \mathbf{S}\left[\mathbf{S}^2\mathbf{x}\right] = \mathbf{S}\left[\mathbf{S}\left(\mathbf{S}\mathbf{x}\right)\right] = \begin{bmatrix} \vdots & \vdots & \vdots & \\ \cdots & 0 & 0 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & 0 & \cdots \\ \cdots & 0 & 0 & 1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

- 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 as a Polynomial on the Line Adjacency

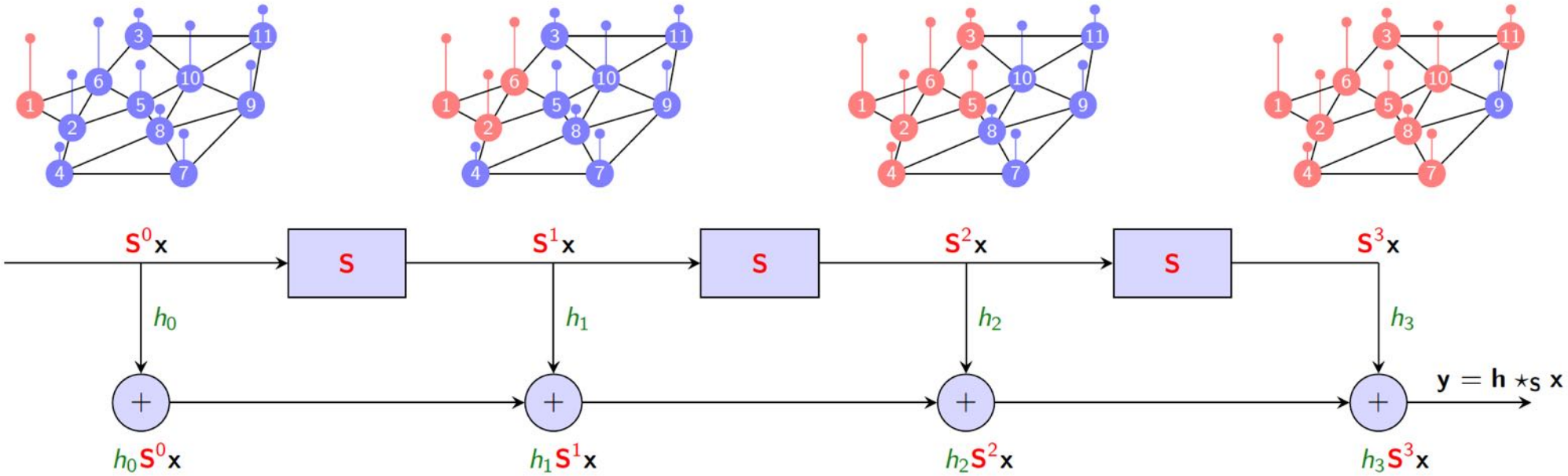
- The convolution operation is a linear combination of shifted versions of the input signal
- But we now know that time shifts are multiplications with the adjacency matrix $\mathbf{S}$ of line graph



- 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 Time Convolution Gneralized to Arbitrary Graphs

- If we let **S** be the shift operator of an arbitrary graph we recover the graph convolution



slide credit: Alejandro Ribeiro

Learning with Graph Signals

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

slide credit: Alejandro Ribeiro

Empirical Risk Minimization

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

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

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

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

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

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

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

slide credit: Alejandro Ribeiro

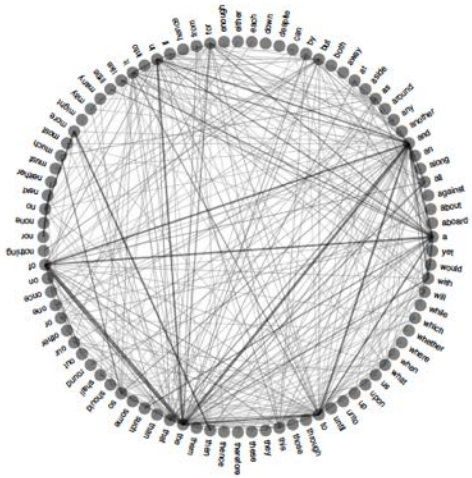
Empirical Risk Minimization with Graph Signals

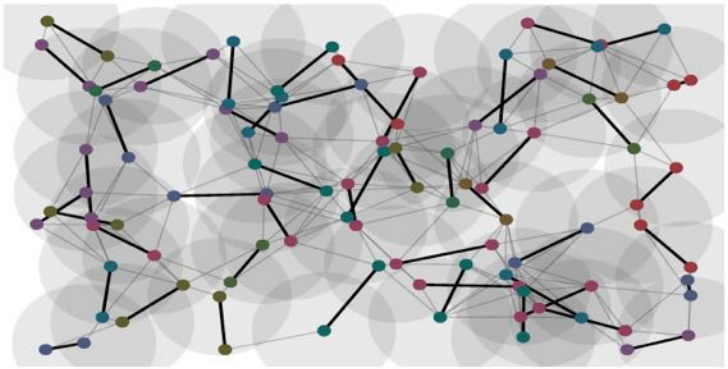
- In ERM, the function class $\mathcal{C}$ is the degree of freedom available to the system's designer

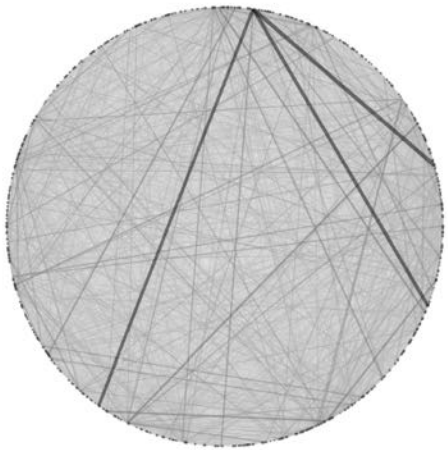
$$\Phi^* = \underset{\Phi \in \mathcal{C}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{y}, \Phi(\mathbf{x})\Big)$$

- 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

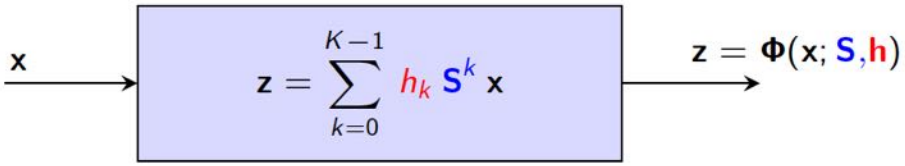






Learning with Graph Convolutional Filters

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



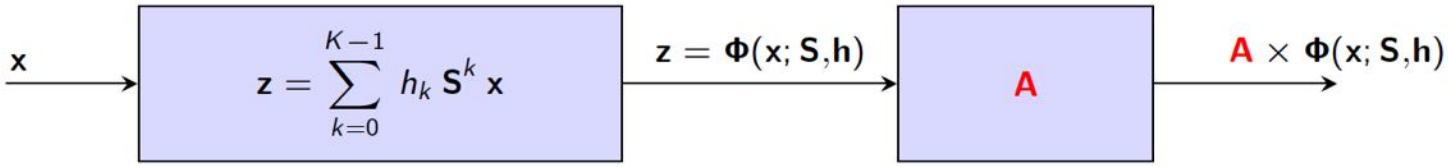
- Learn ERM solution restricted to graph filter class $\Rightarrow$ $\mathbf{h}^* = \underset{\mathbf{h}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{y}, \Phi(\mathbf{x};\mathbf{S},\mathbf{h})\Big)$

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

slide credit: Alejandro Ribeiro

When the Output is Not a Graph Signal: Readout

- Outputs $\mathbf{y} \in \mathbb{R}^m$ are not graph signals $\Rightarrow$ Add readout layer at filter's output to match dimensions
- Readout matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ yields parametrization $\Rightarrow$ $\mathbf{A} \times \Phi(\mathbf{x};\mathbf{S},\mathbf{h}) = \mathbf{A} \times \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$



- Making $\mathbf{A}$ trainable is inadvisable. Learn filter only. $\Rightarrow \mathbf{h}^* = \underset{\mathbf{h}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \ell\Big(\mathbf{y}, \mathbf{A} \times \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h}) \Big)$
- Readouts are simple. Read out node i $\Rightarrow \mathbf{A} = \mathbf{e}_i^T$. Read out signal average $\Rightarrow \mathbf{A} = \mathbf{1}^T$.

slide credit: Alejandro Ribeiro

Graph Neural Networks (GNNs)

slide credit: Alejandro Ribeiro

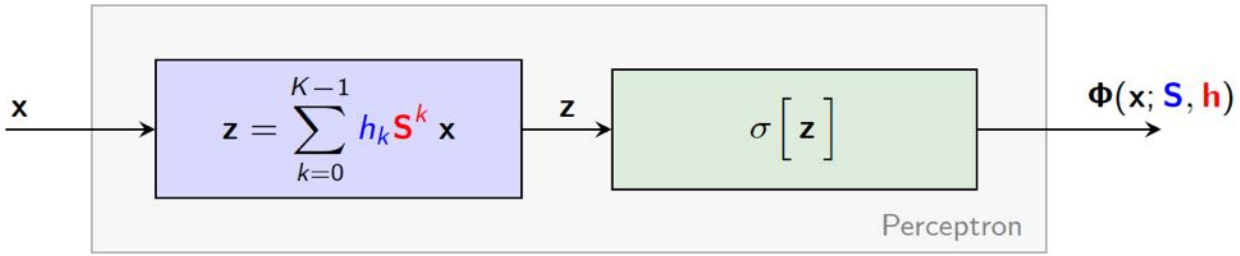
Pointwise Nonlinearity

- 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.

slide credit: Alejandro Ribeiro

Learning with a Graph Perceptron

- Graph filters have limited expressive power because they can only learn linear maps
- A first approach to nonlinear maps is the graph perceptron $\Rightarrow$ $\Phi(\mathbf{x}) = \sigma\left[\sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}\right] = \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h})$



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

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

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

slide credit: Alejandro Ribeiro

Graph Neural Network (GNN)

- 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 $\mathbf{x}_L$

slide credit: Alejandro Ribeiro

The GNN Layer Recursion

- A generic layer of the GNN, Layer ℓ, takes as input the output $\mathbf{x}_{\ell-1}$ of the previous layer $(\ell-1)$
- Layer ℓ processes its input signal $\mathbf{x}_{\ell-1}$ with perceptron $\mathbf{h}_\ell = [h_{\ell 0}, \ldots, h_{\ell,K-1}]$ to produce output $\mathbf{x}_\ell$

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

- With the convention that the Layer 1 input is $\mathbf{x}_0 = \mathbf{x}$, this provides a recursive definition of a GNN
- If it has L layers, the GNN output $\Rightarrow \mathbf{x}_L = \Phi\Big(\mathbf{x}; \mathbf{S}, \mathbf{h}_1, \ldots, \mathbf{h}_L\Big) = \Phi\Big(\mathbf{x}; \mathbf{S}, \mathcal{H}\Big)$
- The filter tensor $\mathcal{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_L]$ is the trainable parameter. The graph shift is prior information

slide credit: Alejandro Ribeiro

slide credit: Alejandro Ribeiro

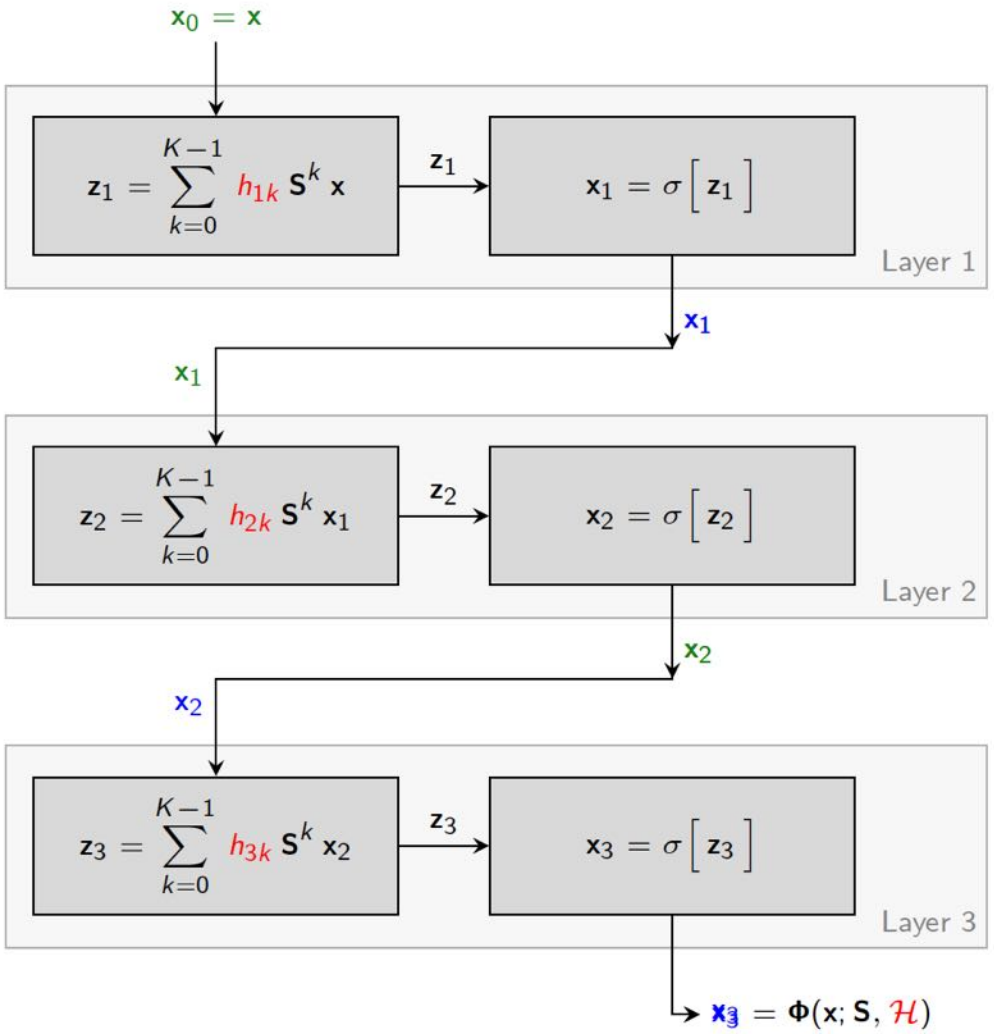
GNN Block Diagram

- Illustrate definition with a GNN with 3 layers
- Feed input signal $\mathbf{x} = \mathbf{x}_0$ into Layer 1

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

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

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



slide credit: Alejandro Ribeiro

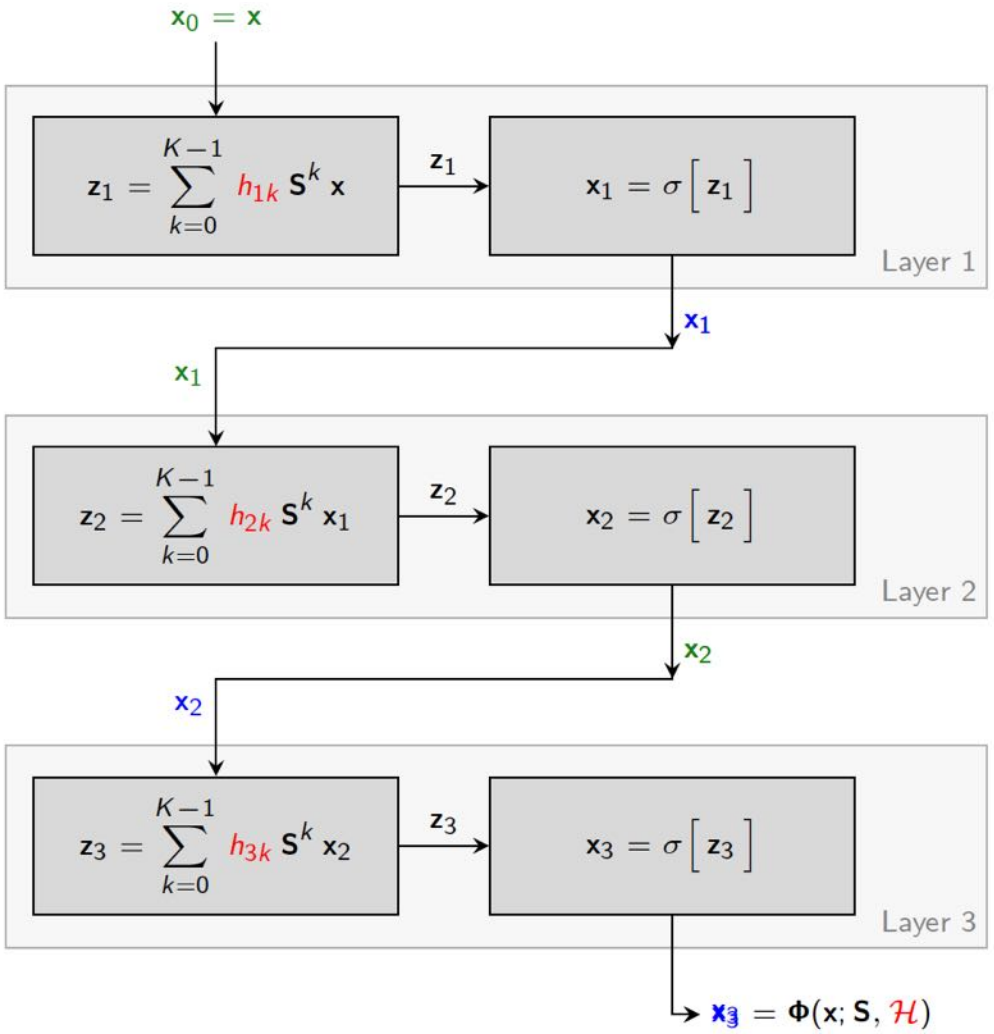
GNN Block Diagram

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

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

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

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



slide credit: Alejandro Ribeiro

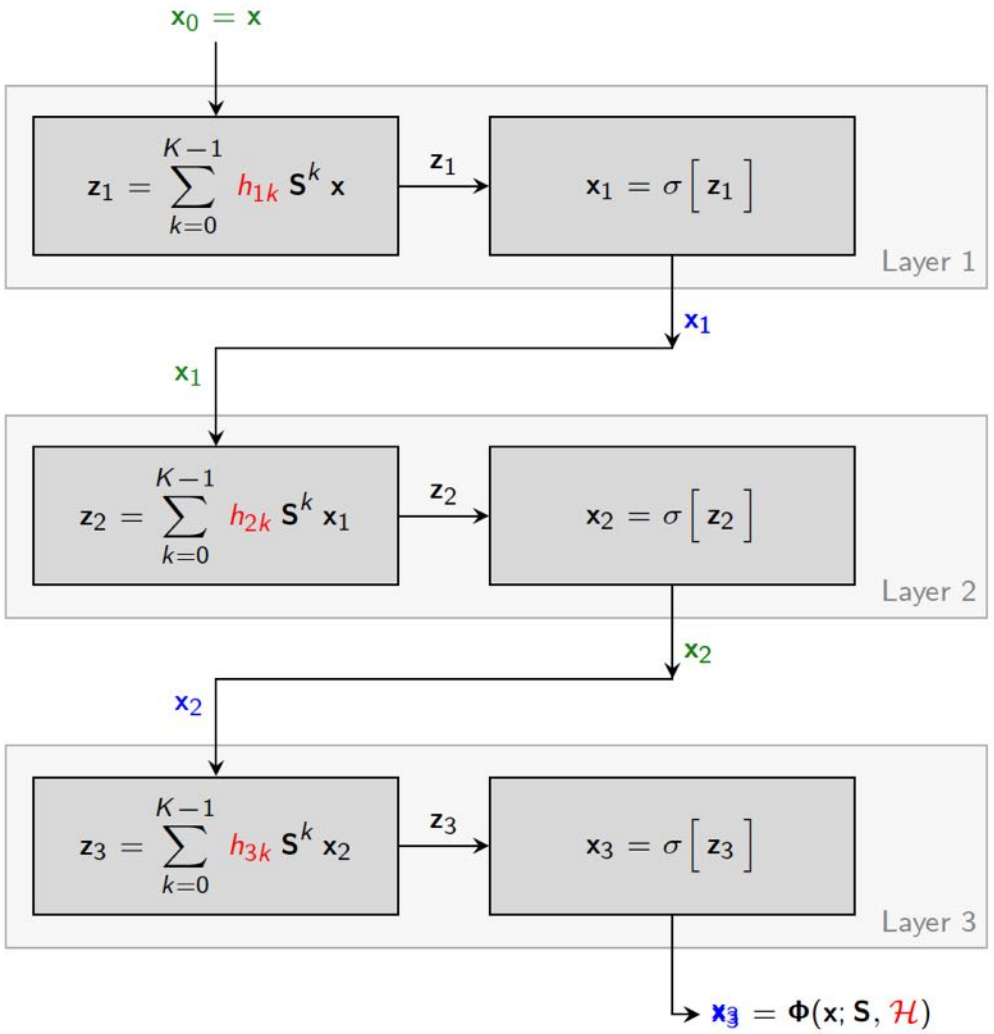
GNN Block Diagram

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

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

- 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

slide credit: Alejandro Ribeiro

slide credit: Alejandro Ribeiro

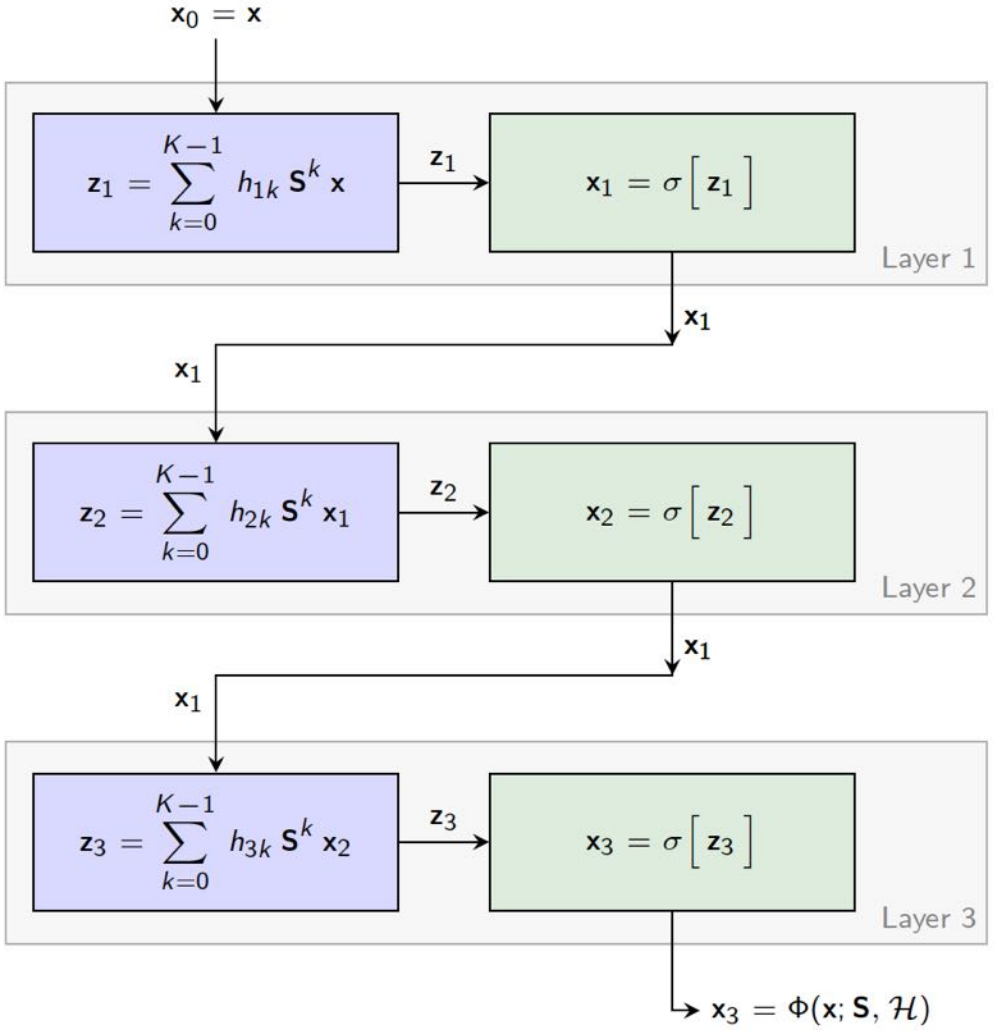
Components of a Graph Neural Network

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

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

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

$\Rightarrow$ Compositions of Filters & Pointwise nonlinearities



slide credit: Alejandro Ribeiro

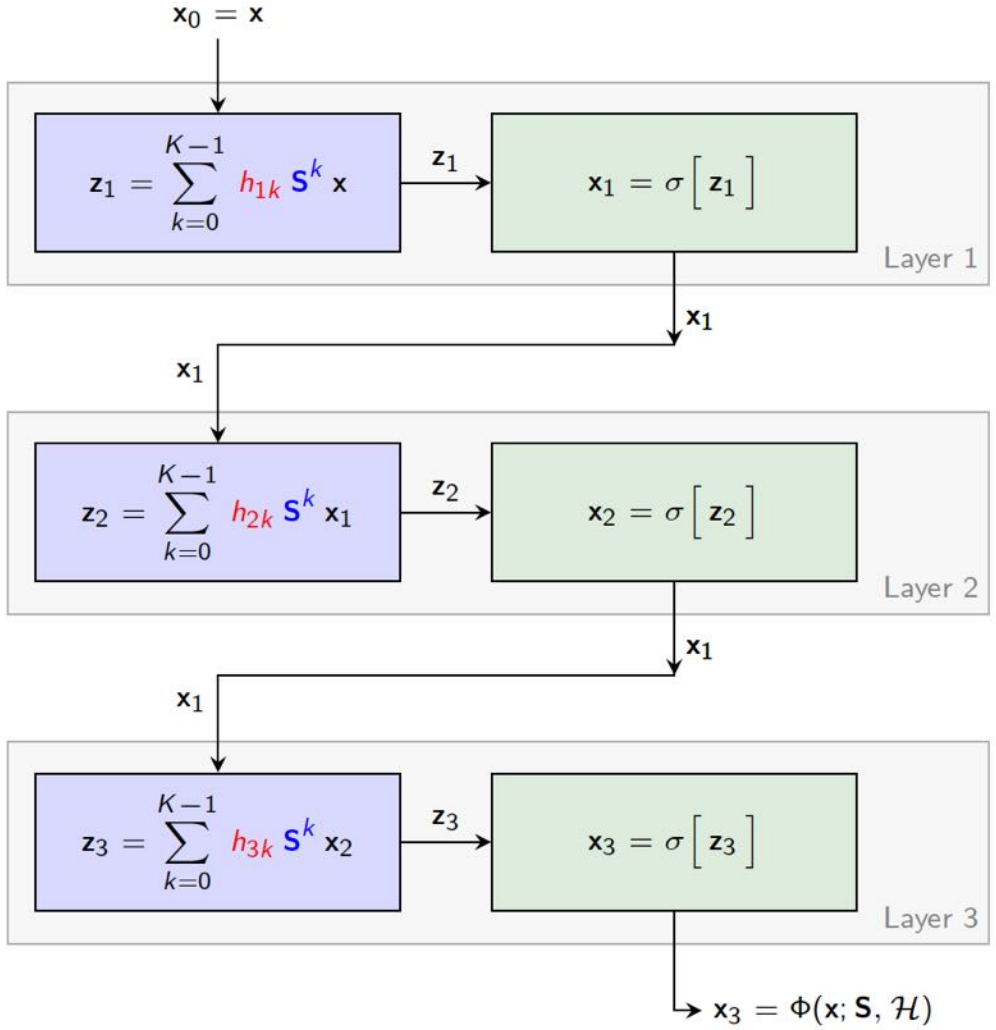
Components of a Graph Neural Network

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

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

- Filters are parametrized by...

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



slide credit: Alejandro Ribeiro

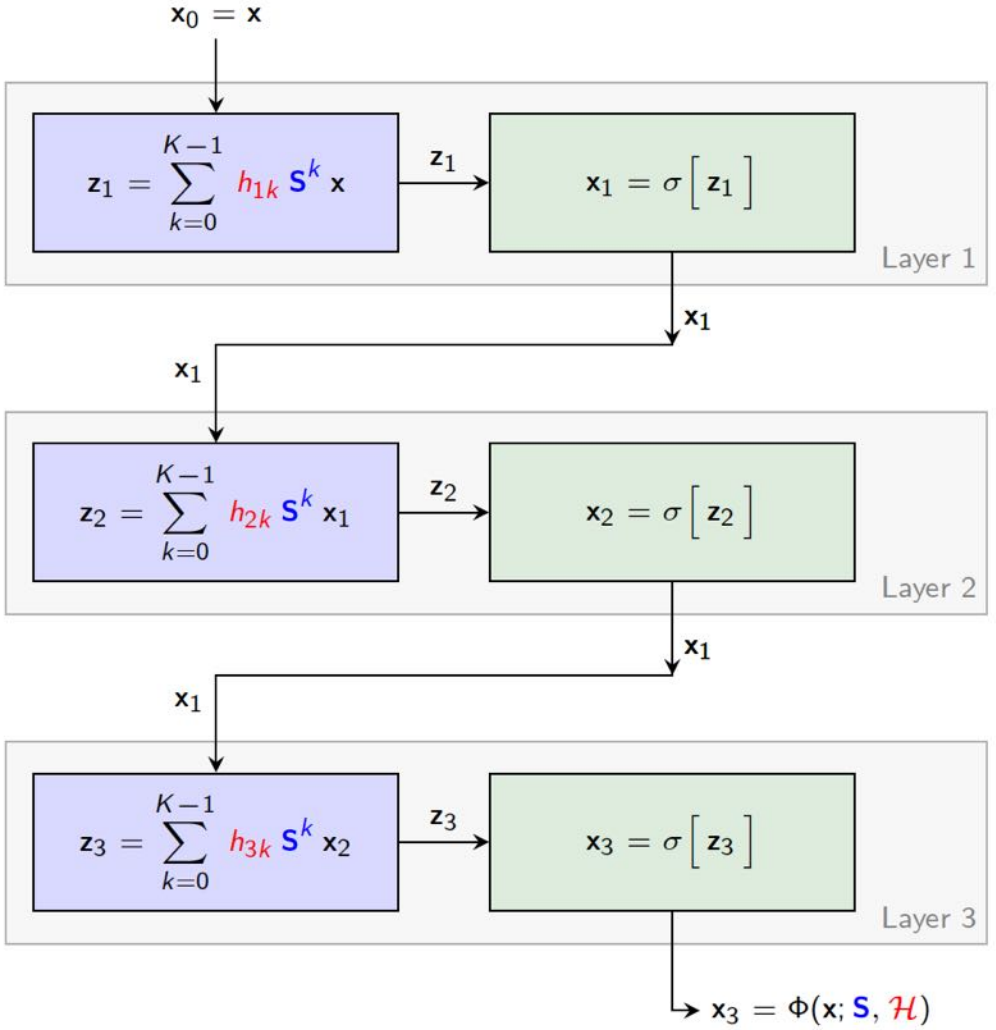
Components of a Graph Neural Network

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

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

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

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



slide credit: Alejandro Ribeiro

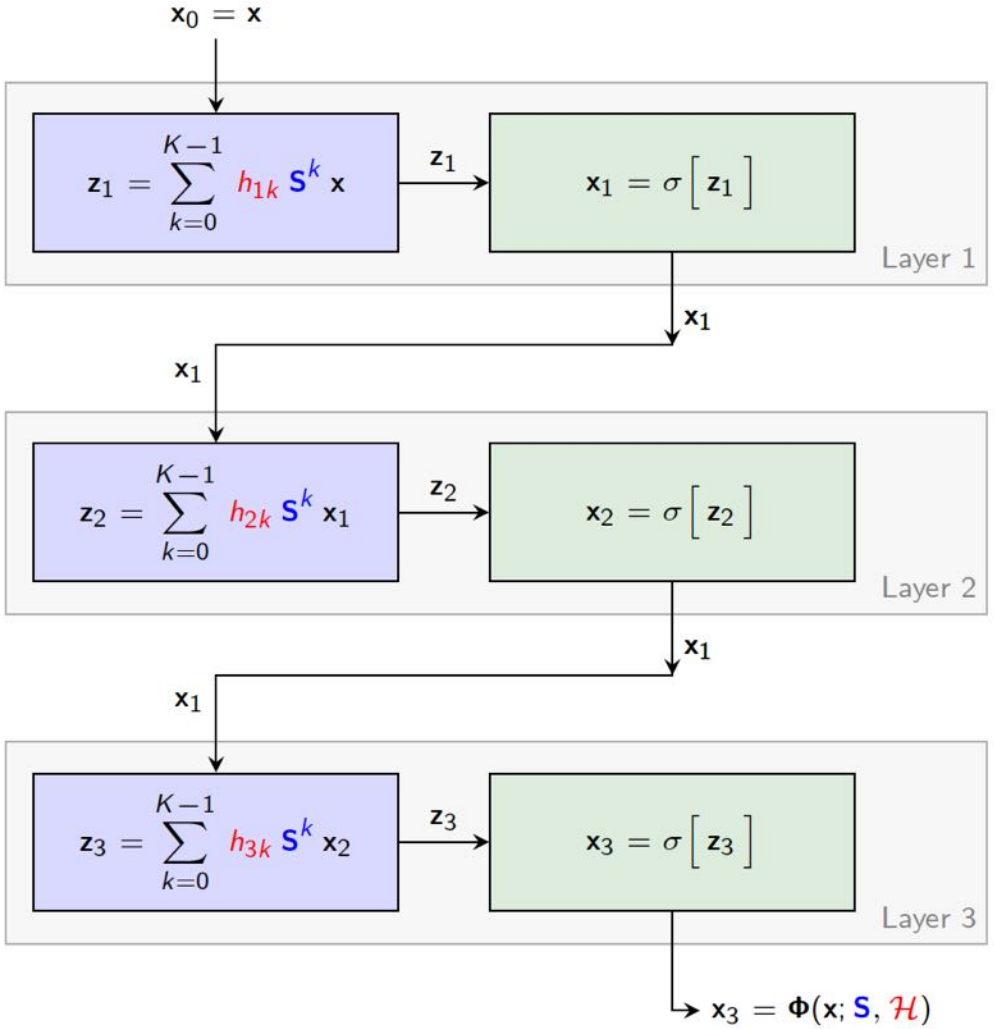
Learning a Graph Neural Network

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

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

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

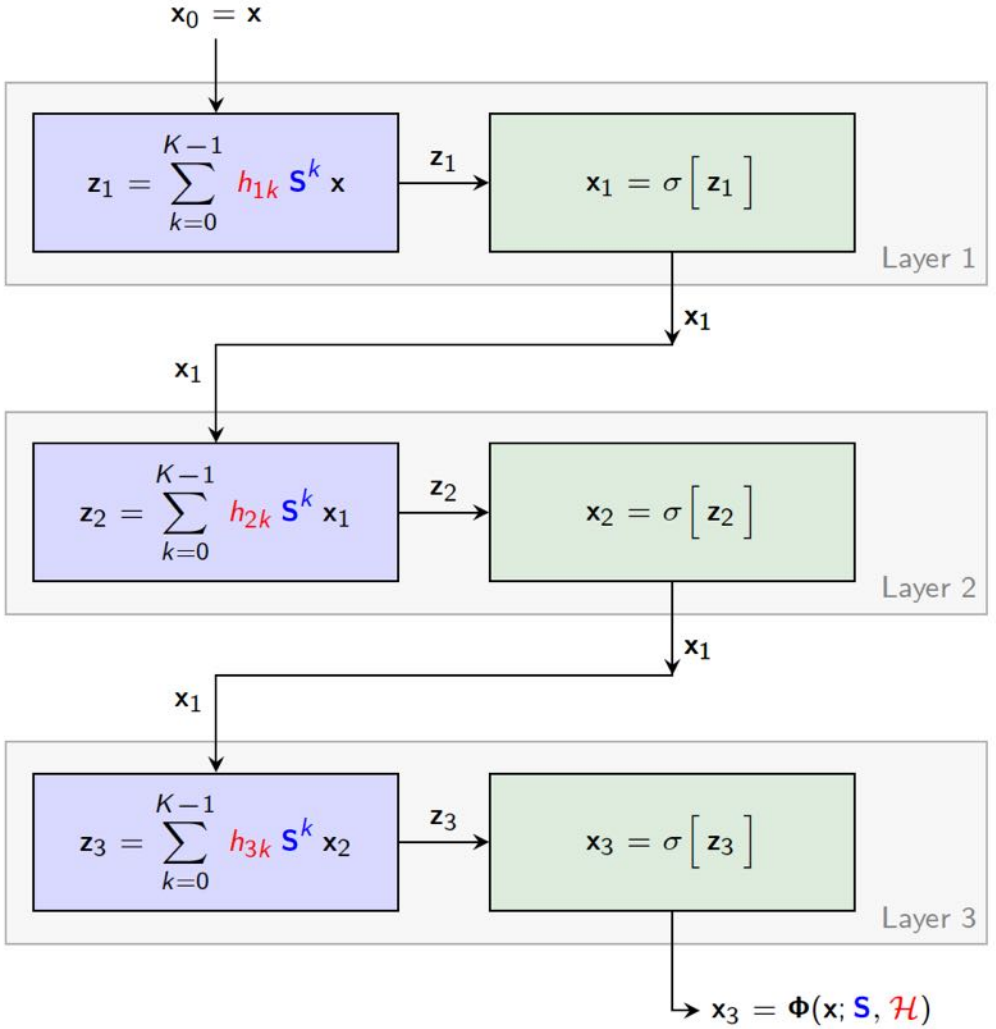
  $\Rightarrow$ Prior information given to the GNN



slide credit: Alejandro Ribeiro

Graph Neural Networks and Graph Filters

- GNNs are minor variations of graph filters
- Add pointwise nonlinearities and layer compositions
  - $\Rightarrow$ Nonlinearities process individual entries
  - $\Rightarrow$ Component mixing is done by graph filters only
- GNNs do work (much) better than graph filters

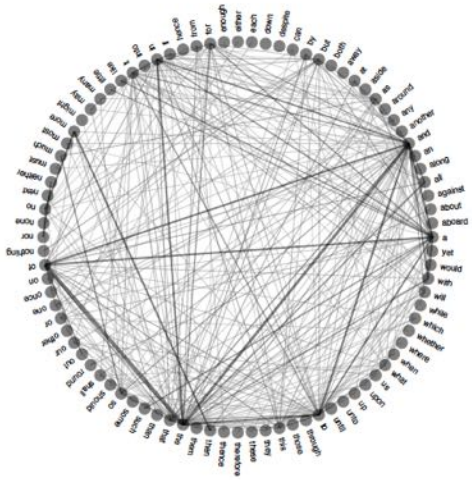


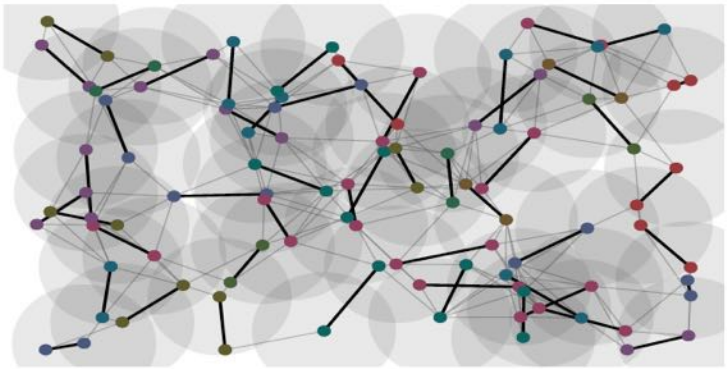
Fully Connected Neural Networks

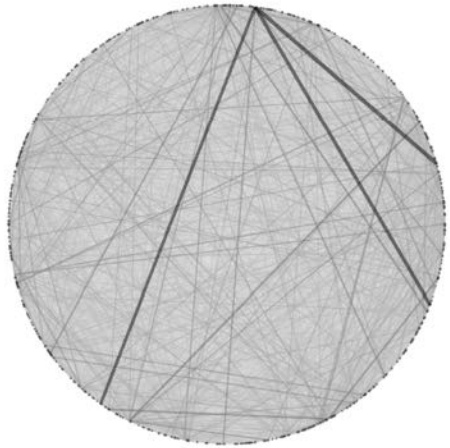
slide credit: Alejandro Ribeiro

The Road not Taken: Fully Convolutional 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



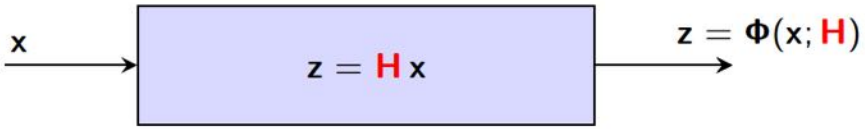




slide credit: Alejandro Ribeiro

Learning with a Linear Classifier

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

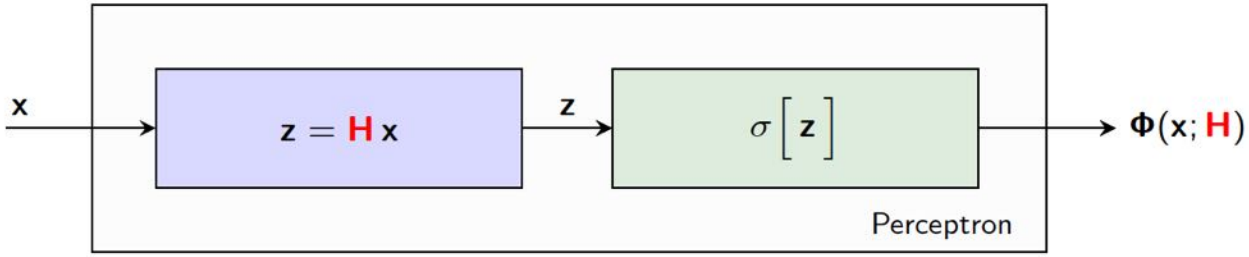


- Optimal regressor is ERM solution restricted to linear class $\Rightarrow \mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{\Phi}(\mathbf{x}; \mathbf{H}), \mathbf{y}\Big)$

slide credit: Alejandro Ribeiro

Learning with a Linear Perceptron

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



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

slide credit: Alejandro Ribeiro

Fully Connected Neural Network

- 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.

slide credit: Alejandro Ribeiro

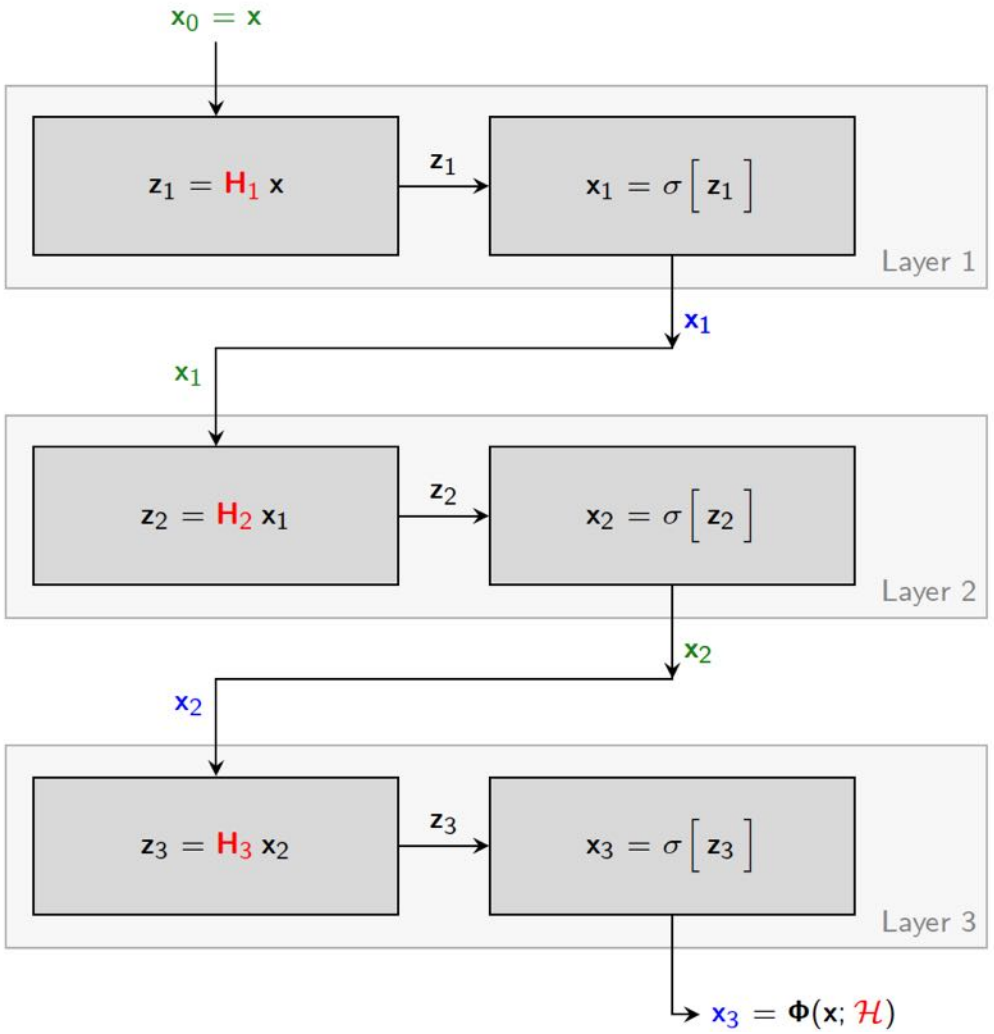
slide credit: Alejandro Ribeiro

Fully Connected Neural Network

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



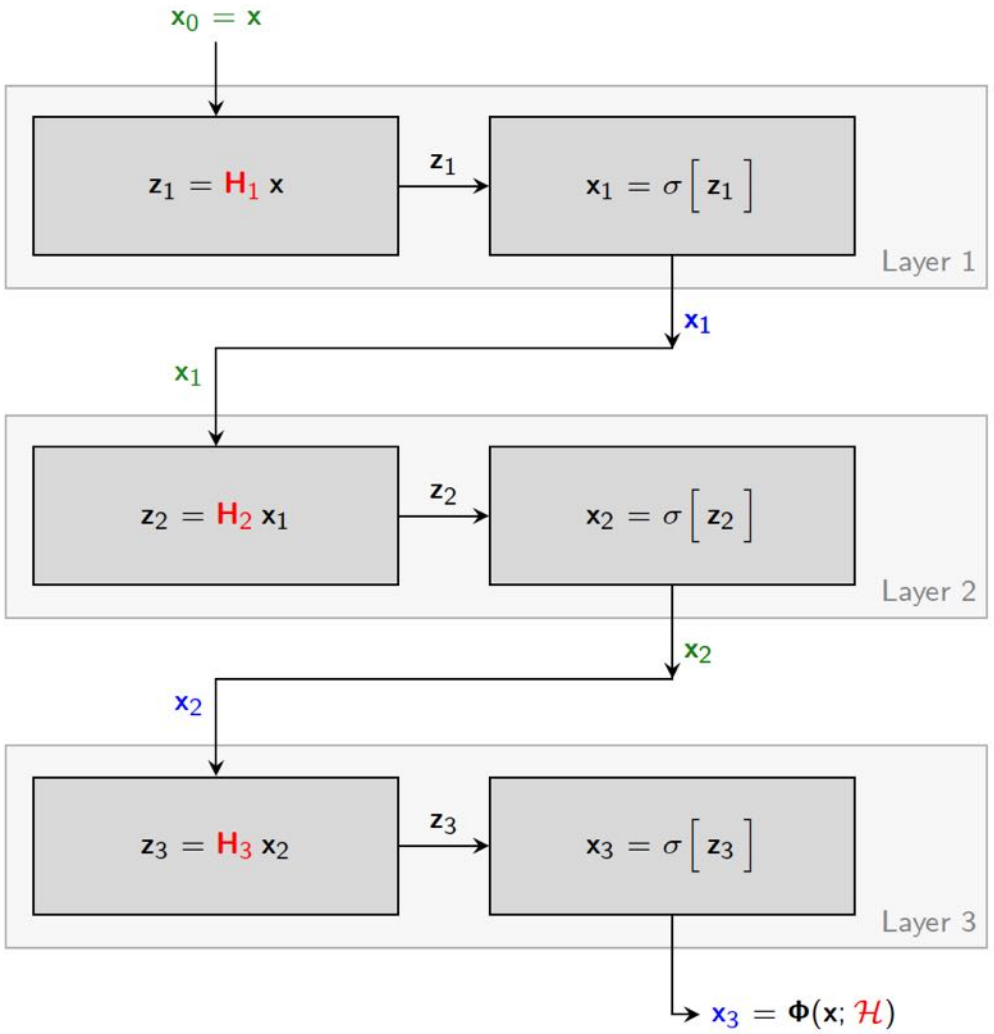
slide credit: Alejandro Ribeiro

Fully Connected Neural Network

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

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

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



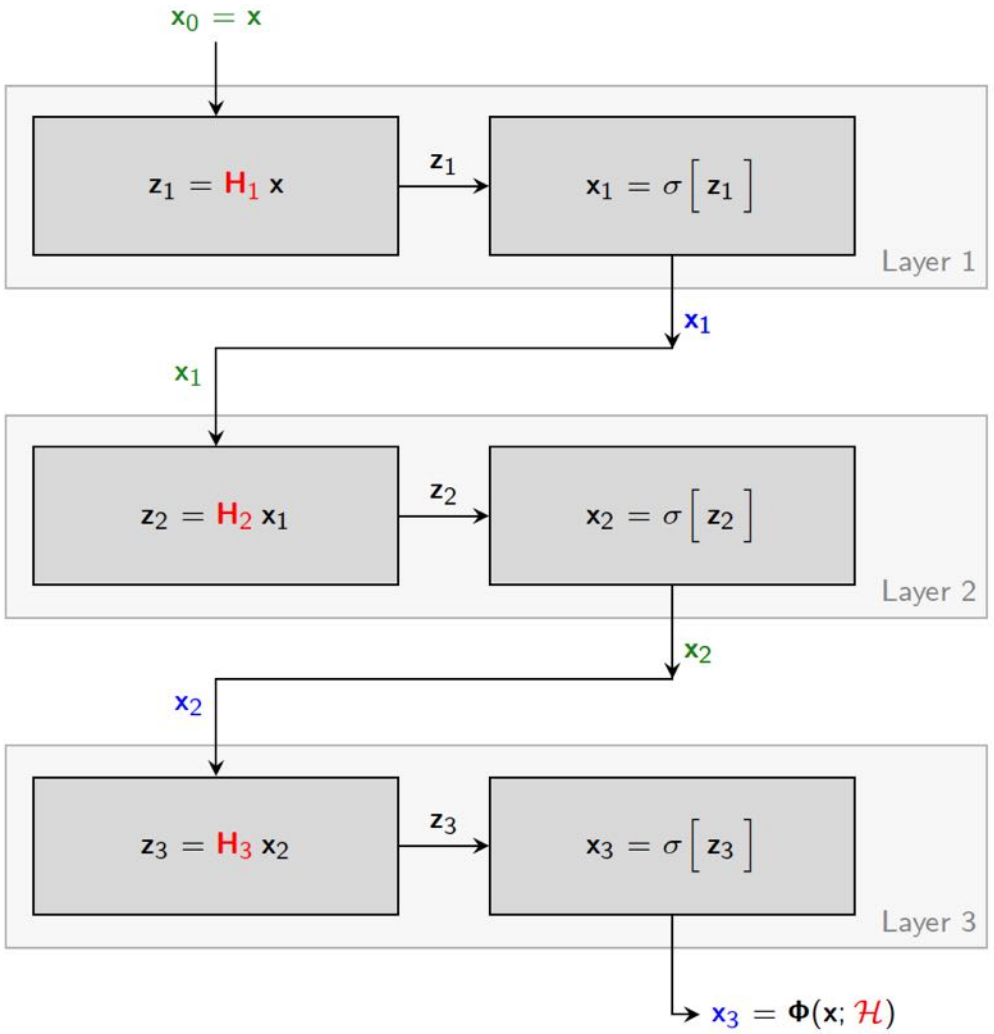
slide credit: Alejandro Ribeiro

Fully Connected Neural Network

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

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

- 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

slide credit: Alejandro Ribeiro

Which is Better: Graph NN or 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

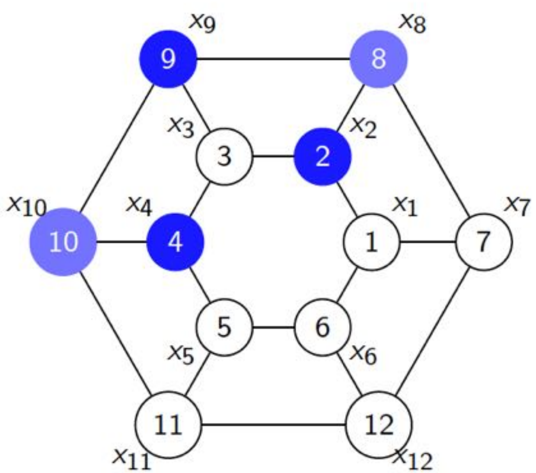
- In practice, the GNN does better because it generalizes better to unseen signals

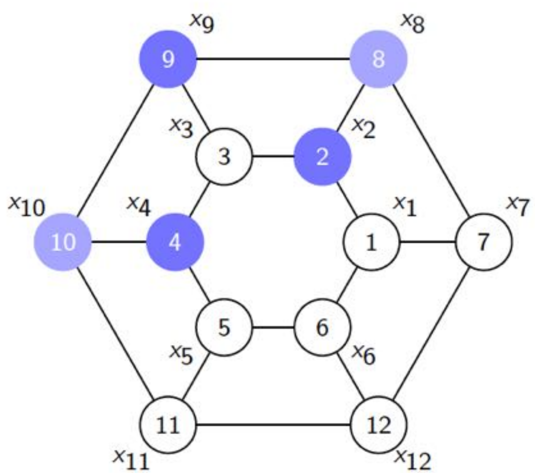
  $\Rightarrow$ Because it exploits internal symmetries of graph signals codified in the graph shift operator

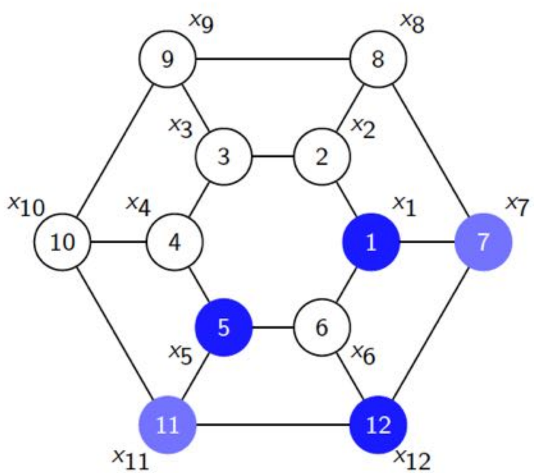
slide credit: Alejandro Ribeiro

Generalization with a Neural Network

- 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



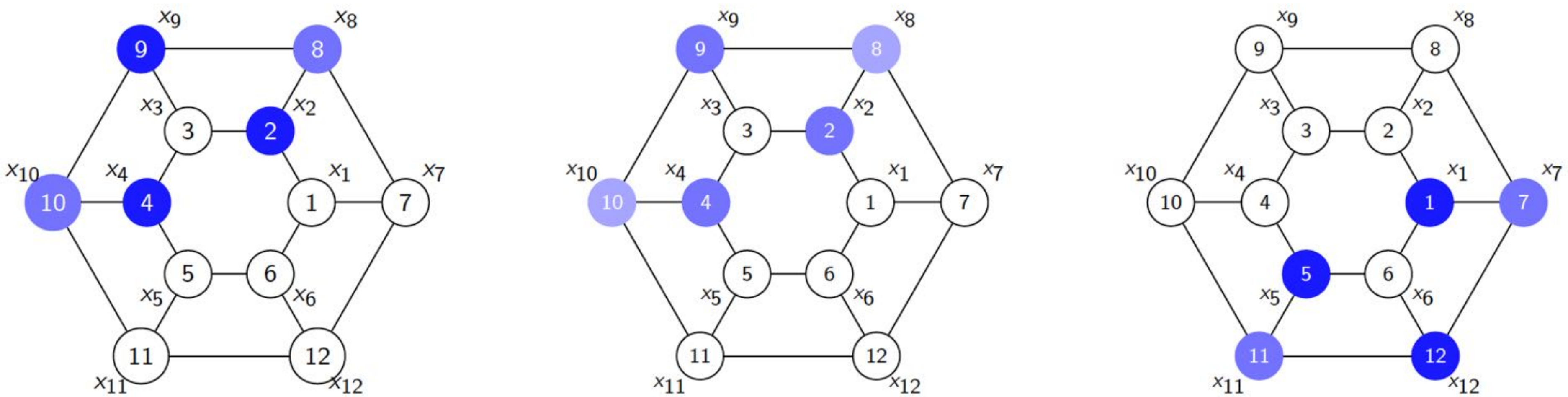




slide credit: Alejandro Ribeiro

Generalization with a Graph Neural Network

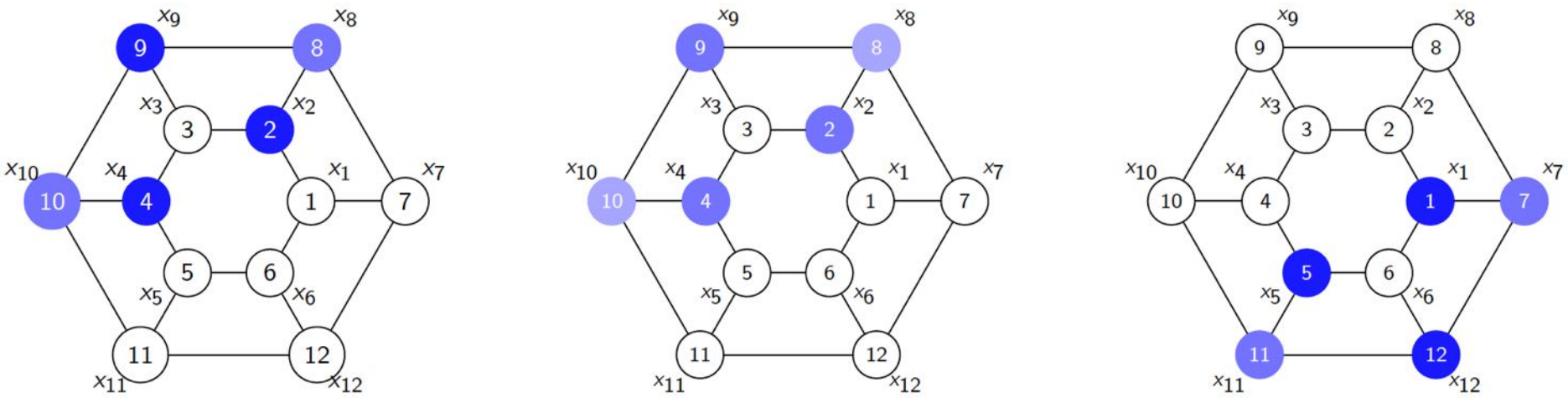
- The GNN will succeed at predicting ratings for the signal on the right because it knows the graph
- The GNN still learns how to fill the middle signal. But it also learns how to fill the right signal



slide credit: Alejandro Ribeiro

Permutation Equivariance of GNNs

- 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



slide credit: Alejandro Ribeiro

Permutation Equivariance of Graph Filters

- We will show that graph convolutional filters are equivariant to permutations

slide credit: Alejandro Ribeiro

Permutation Matrices

Definition (Permutation matrix)

A square matrix $\mathbf{P}$ is a permutation matrix if it has binary entries so that $\mathbf{P} \in \{0,1\}^{n \times n}$

and it further satisfies $\mathbf{P1} = \mathbf{1}$ and $\mathbf{P}^T\mathbf{1} = \mathbf{1}$.

- The product $\mathbf{P}^T\mathbf{x}$ reorders the entries of the vector $\mathbf{x}$.
- The product $\mathbf{P}^T\mathbf{SP}$ is a consistent reordering of the rows and columns of $\mathbf{S}$

slide credit: Alejandro Ribeiro

Permutation Matrices

Definition (Permutation matrix)

A square matrix $\mathbf{P}$ is a permutation matrix if it has binary entries so that $\mathbf{P} \in \{0,1\}^{n \times n}$

and it further satisfies $\mathbf{P1} = \mathbf{1}$ and $\mathbf{P}^T\mathbf{1} = \mathbf{1}$.

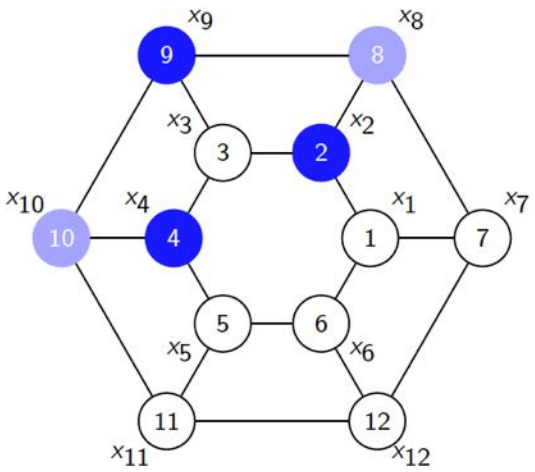
- Since $\mathbf{P1} = \mathbf{P}^T\mathbf{1} = \mathbf{1}$ with binary entries $\Rightarrow$ Exactly one nonzero entry per row and column of $\mathbf{P}$
- Permutation matrices are unitary $\Rightarrow \mathbf{P}^T\mathbf{P} = \mathbf{I}$. Matrix $\mathbf{P}^T$ undoes the reordering of matrix $\mathbf{P}$

slide credit: Alejandro Ribeiro

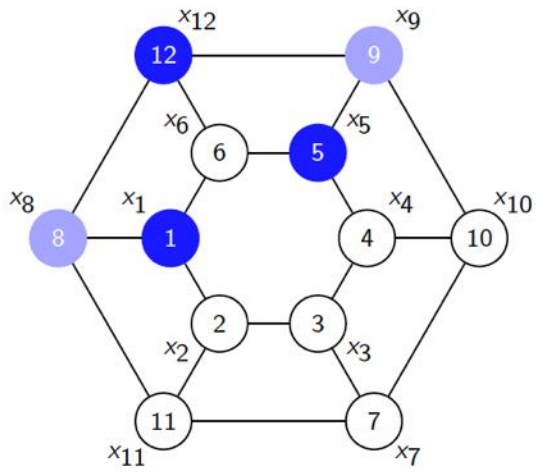
Relabelling of Graph Signals

- If $(\mathbf{S}, \mathbf{x})$ is a graph signal, $(\mathbf{P}^T\mathbf{S}\mathbf{P}, \mathbf{P}^T\mathbf{x})$ is a relabeling of $(\mathbf{S}, \mathbf{x})$. Same signal. Different names

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



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



- Processing should be label-independent $\Rightarrow$ Permutation equivariance of graph filters and GNNs

slide credit: Alejandro Ribeiro

Graph Filters and the Permutation of Graph Signals

- Graph filter $\mathbf{H}(\mathbf{S})$ is a polynomial on shift operator $\mathbf{S}$ with coefficients h_k. Outputs given by

$$\mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$$

- We consider running the same filter on $(\mathbf{S}, \mathbf{x})$ and permuted (relabeled) $(\hat{\mathbf{S}}, \hat{\mathbf{x}}) = (\mathbf{P}^T\mathbf{S}\mathbf{P}, \mathbf{P}^T\mathbf{x})$

$$\mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x} \qquad \mathbf{H}(\hat{\mathbf{S}})\hat{\mathbf{x}} = \sum_{k=0}^{K-1} h_k \hat{\mathbf{S}}^k \hat{\mathbf{x}}$$

- Filter $\mathbf{H}(\mathbf{S})\mathbf{x}$ $\Rightarrow$ Coefficients h_k. Input signal $\mathbf{x}$. Instantiated on shift $\mathbf{S}$
- Filter $\mathbf{H}(\hat{\mathbf{S}})\hat{\mathbf{x}}$ $\Rightarrow$ Same Coefficients h_k. Permuted Input signal $\hat{\mathbf{x}}$. Instantiated on permuted shift $\hat{\mathbf{S}}$

slide credit: Alejandro Ribeiro

Permutation Equivariance of Graph Filters

Theorem (Permutation equivariance of graph filters)

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

$$\mathbf{H}(\hat{\mathbf{S}})\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{H}(\mathbf{S})\mathbf{x}$$

- Graph filters are equivariant to permutations $\Rightarrow$ Permute input and shift $\equiv$ Permute output

slide credit: Alejandro Ribeiro

Proof of Permutation Equivariance of Graph Filters

Proof: Write filter output in polynomial form. Use permutation definitions $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{S}\mathbf{P}$ and $\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{x}$

$$\mathbf{H}(\hat{\mathbf{S}})\hat{\mathbf{x}} = \sum_{k=0}^{K-1} h_k \hat{\mathbf{S}}^k \hat{\mathbf{x}} = \sum_{k=0}^{K-1} h_k \left(\mathbf{P}^T\mathbf{S}\mathbf{P}\right)^k \mathbf{P}^T\mathbf{x}$$

- In the powers $\left(\mathbf{P}^T\mathbf{S}\mathbf{P}\right)^k$, $\mathbf{P}$ and $\mathbf{P}^T$ undo each other $(\mathbf{P}^T\mathbf{P} = \mathbf{I}) \Rightarrow \left(\mathbf{P}^T\mathbf{S}\mathbf{P}\right)^k = \mathbf{P}^T\left(\mathbf{S}\right)^k\mathbf{P}$
- Substitute this into filter's output expression. Cancel remaining $\mathbf{P}\mathbf{P}^T = \mathbf{I}$ product. Factor $\mathbf{P}^T$

$$\mathbf{H}(\hat{\mathbf{S}})\hat{\mathbf{x}} = \sum_{k=0}^{K-1} h_k \mathbf{P}^T\mathbf{S}^k\mathbf{P}\mathbf{P}^T\mathbf{x} = \sum_{k=0}^{K-1} h_k \mathbf{P}^T\mathbf{S}^k\mathbf{I}\mathbf{x} = \mathbf{P}^T\sum_{k=0}^{K-1} h_k \mathbf{S}^k\mathbf{x} = \mathbf{P}^T\mathbf{H}(\mathbf{S})\mathbf{x} \qquad \blacksquare$$

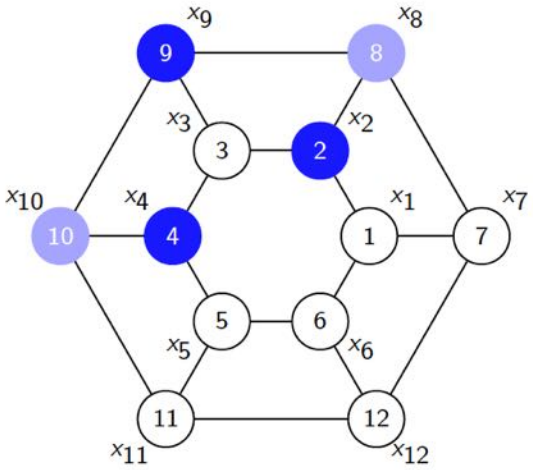
slide credit: Alejandro Ribeiro

Graph Filter Processing is Independent of Graph Labeling

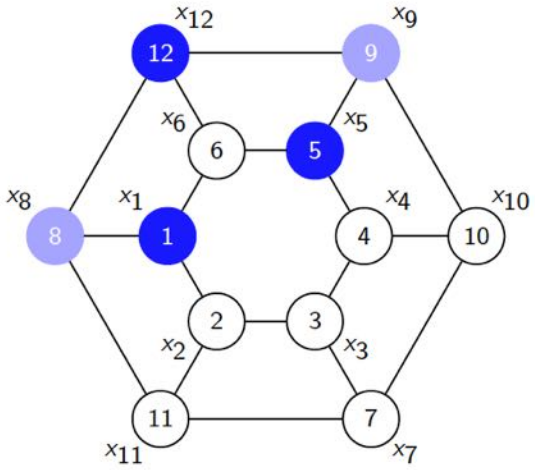
- We requested signal processing independent of labeling $\Rightarrow$ Graph filters fulfill this request

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

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



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



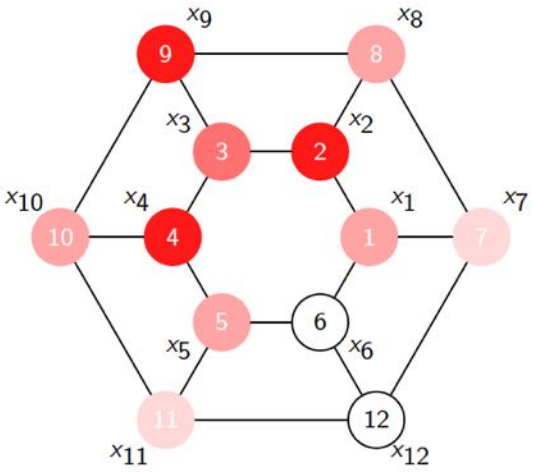
slide credit: Alejandro Ribeiro

Graph Filter Processing is Independent of Graph Labeling

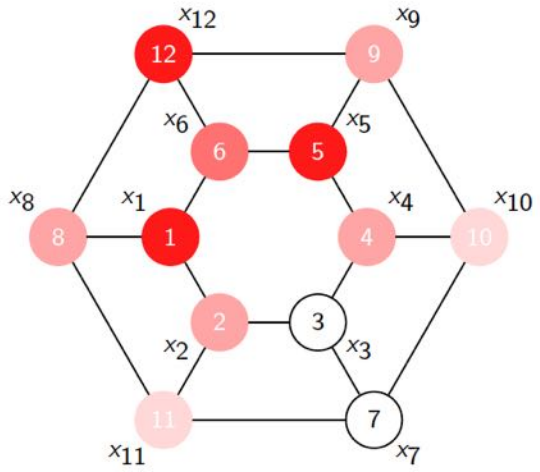
- We requested signal processing independent of labeling $\Rightarrow$ Graph filters fulfill this request

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

Filter's output $\mathbf{H(S)x}$ Supported on $\mathbf{S}$



Filter's Output $\mathbf{H(\hat{S})\hat{x}}$ supported on $\hat{\mathbf{S}}$



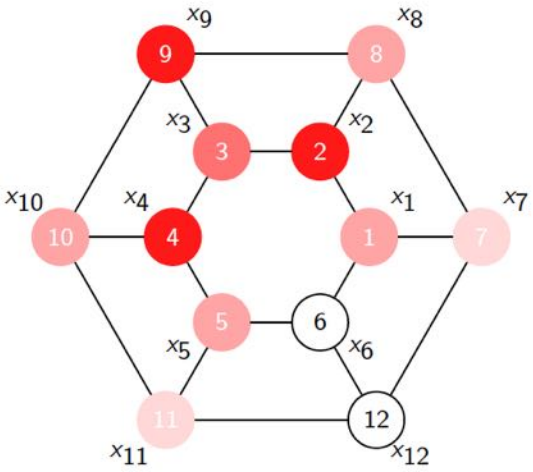
slide credit: Alejandro Ribeiro

Graph Filter Processing is Independent of Graph Labeling

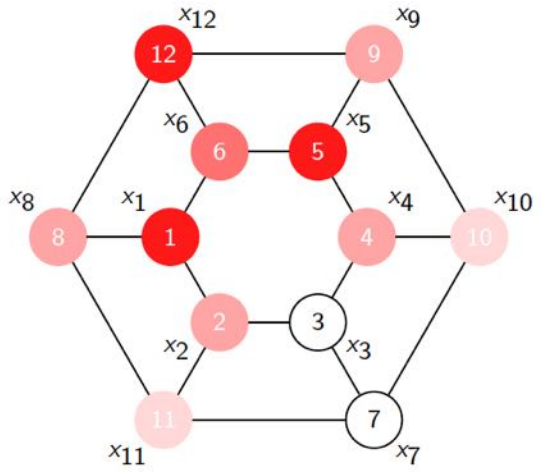
- We requested signal processing independent of labeling $\Rightarrow$ Graph filters fulfill this request

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

Filter's output $\mathbf{H(S)x}$ Supported on $\mathbf{S}$



Equivariance theorem $\Rightarrow \mathbf{H}(\hat{\mathbf{S}})\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{H}(\mathbf{S})\mathbf{x}$



slide credit: Alejandro Ribeiro

Permutation Equivariance of Graph Neural Networks

- We will show that graph neural networks inherit the permutation equivariance of graph filters

slide credit: Alejandro Ribeiro

GNNs and Permutations of Graph Signals

- L layers recursively process outputs of previous layers. GNN Output parametrized by tensor $\mathcal{H}$

$$\mathbf{x}_\ell = \sigma\left[\sum_{k=0}^{K-1} h_{\ell k}\, \mathbf{S}^k\, \mathbf{x}_{\ell-1}\right] = \sigma\left[\mathbf{H}_\ell(\mathbf{S})\, \mathbf{x}_{\ell-1}\right] \qquad \Phi\Big(\mathbf{x};\ \mathbf{S},\ \mathcal{H}\Big) = \mathbf{x}_L$$

- We consider running the same GNN on $(\mathbf{S}, \mathbf{x})$ and permuted (relabeled) $(\hat{\mathbf{S}}, \hat{\mathbf{x}}) = (\mathbf{P}^T\mathbf{S}\mathbf{P}, \mathbf{P}^T\mathbf{x})$

$$\Phi\Big(\mathbf{x};\ \mathbf{S},\ \mathcal{H}\Big) \qquad \Phi\Big(\hat{\mathbf{x}};\ \hat{\mathbf{S}},\ \mathcal{H}\Big)$$

- GNN $\Phi(\mathbf{x}; \mathbf{S}, \mathcal{H}) \Rightarrow$ Tensor $\mathcal{H}$. Input signal $\mathbf{x}$. Instantiated on shift $\mathbf{S}$
- GNN $\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{H}) \Rightarrow$ Same Tensor $\mathcal{H}$. Permuted Input signal $\hat{\mathbf{x}}$. Instantiated on permuted shift $\hat{\mathbf{S}}$

slide credit: Alejandro Ribeiro

Theorem (Permutation equivariance of graph neural networks)

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

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

- GNNs equivariant to permutations $\Rightarrow$ Permute input and shift $\equiv$ Permute output

slide credit: Alejandro Ribeiro

Proof of Permutation Equivariance of GNNs

Proof: GNN Layer ℓ recursion on signal $\mathbf{x}_{\ell-1}$ and shift $\mathbf{S}$ $\Rightarrow \mathbf{x}_\ell = \sigma\left[\sum_{k=0}^{K-1} h_{\ell k}\, \mathbf{S}^k\, \mathbf{x}_{\ell-1}\right] = \sigma\left[\mathbf{H}_\ell(\mathbf{S})\mathbf{x}_{\ell-1}\right]$

GNN Layer ℓ recursion on signal $\hat{\mathbf{x}}_{\ell-1}$ and shift $\hat{\mathbf{S}}$ $\Rightarrow \hat{\mathbf{x}}_\ell = \sigma\left[\sum_{k=0}^{K-1} h_{\ell k}\, \hat{\mathbf{S}}^k\, \hat{\mathbf{x}}_{\ell-1}\right] = \sigma\left[\mathbf{H}_\ell(\hat{\mathbf{S}})\hat{\mathbf{x}}_{\ell-1}\right]$

- Assume Layer ℓ inputs satisfy $\hat{\mathbf{x}}_{\ell-1} = \mathbf{P}^T\mathbf{x}_{\ell-1}$. Filters are equivariant. Linearity is pointwise

$$\hat{\mathbf{x}}_\ell = \sigma\left[\mathbf{H}_\ell(\hat{\mathbf{S}})\hat{\mathbf{x}}_{\ell-1}\right] = \sigma\left[\mathbf{P}^T\mathbf{H}_\ell(\mathbf{S})\mathbf{x}_{\ell-1}\right] = \mathbf{P}^T\sigma\left[\mathbf{H}_\ell(\mathbf{S})\mathbf{x}_{\ell-1}\right] = \mathbf{P}^T\mathbf{x}_\ell$$

- This in an induction step At Layer 1 we have $\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{x}$ by hypothesis. Induction is complete. ■

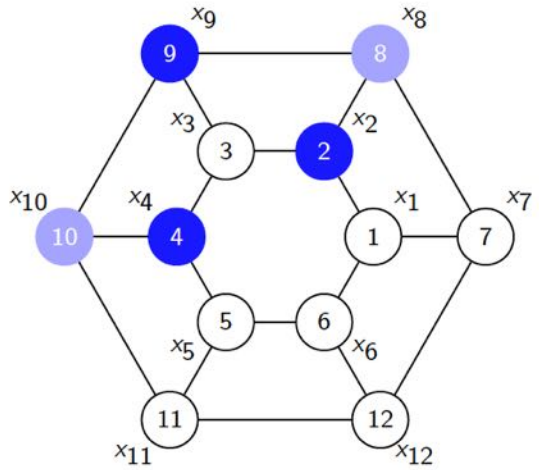
slide credit: Alejandro Ribeiro

GNNs Processing is Independent of Labeling

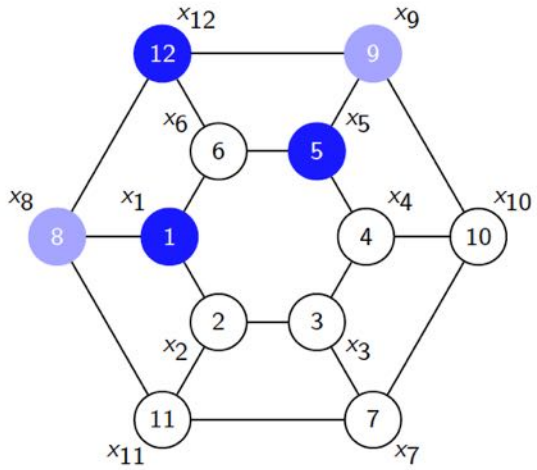
- GNNs, same as graph filters, perform label-independent processing. The nonlinearity is pointwise

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

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



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



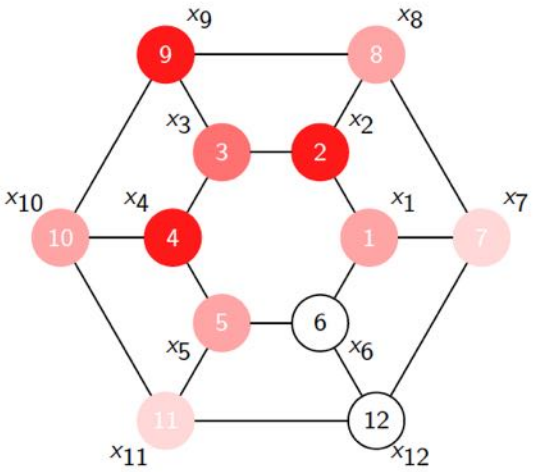
slide credit: Alejandro Ribeiro

GNNs Processing is Independent of Labeling

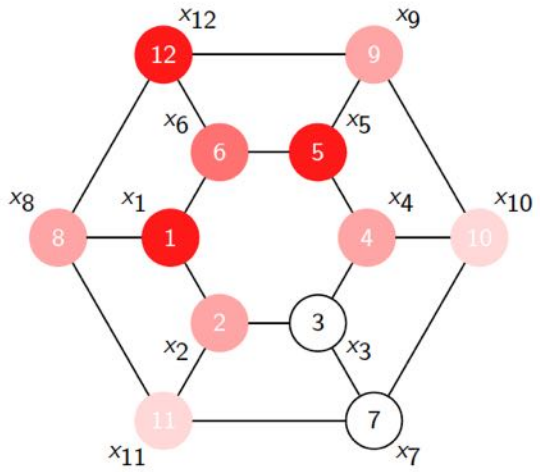
- GNNs, same as graph filters, perform label-independent processing. The nonlinearity is pointwise

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

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



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

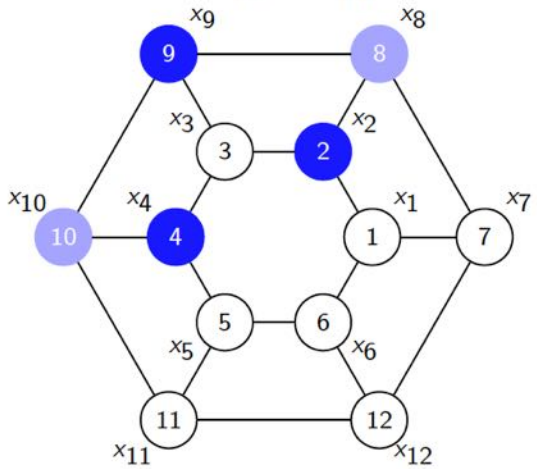


slide credit: Alejandro Ribeiro

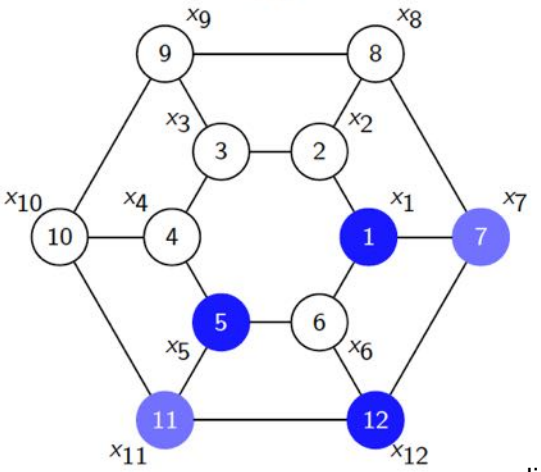
Equivariance to Permutations and Signal Symmetries

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

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



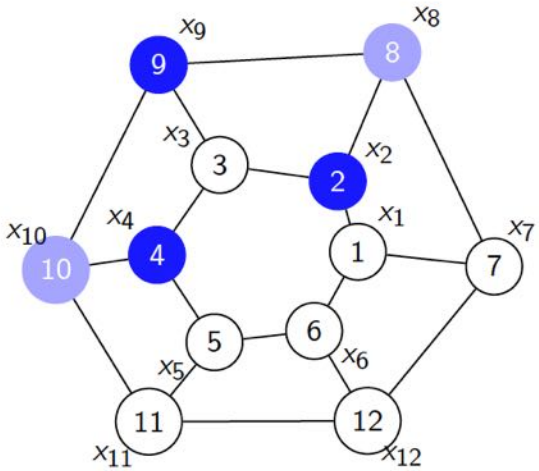
Learn to process $\mathbf{P}^T \mathbf{x}$ supported on $\mathbf{S} = \mathbf{P}^T \mathbf{S} \mathbf{P}$

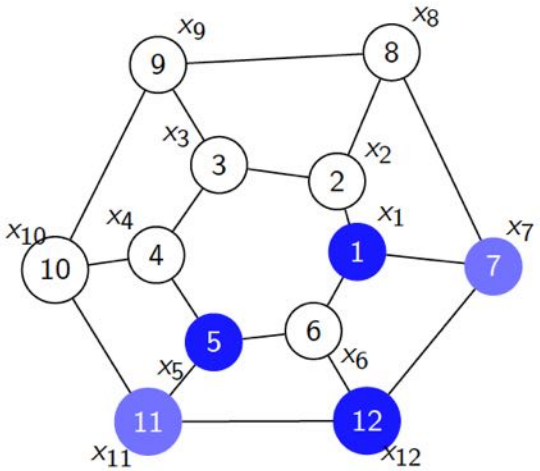


slide credit: Alejandro Ribeiro

Symmetry is Rare but Quasi-Symmetry is Common

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





slide credit: Alejandro Ribeiro