Deep Learning

Lecture 10 – Graph Neural Networks

Kumar Bipin

BE, MS, PhD (MMMTU, IISc, IIIT-Hyderabad)

Robotics, Computer Vision, Deep Learning, Machine Learning, System Software

Agenda

10.1 Machine Learning on Graphs

10.2 Graph Convolution Filters

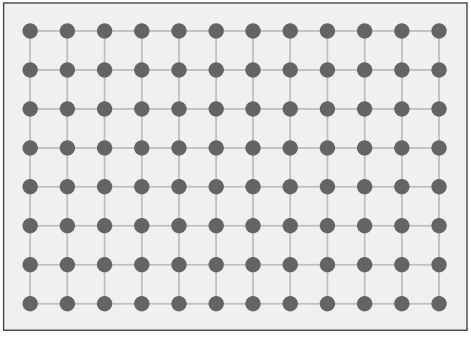
10.3 Graph Convolution Networks

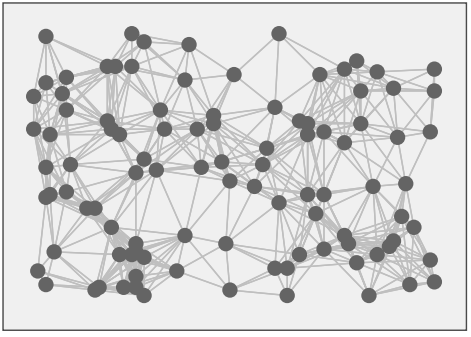
10.1
Machine Learning on Graphs

Motivation

- Multi-Layer Perceptrons (MLPs) are very flexible function approximators
- However, they don't scale (e.g., with the input dimensionality)
- The number of parameters quickly explodes, the model overfits
- Convolutional Neural Networks (CNNs) address this issue
- However, they only work for regularly grid-structured data (e.g., images)
- A lot of data cannot be described in such a regular format (e.g., molecules)
- We seek a model class that:
  1. scales better than MLPs
  2. is more flexible than CNNs

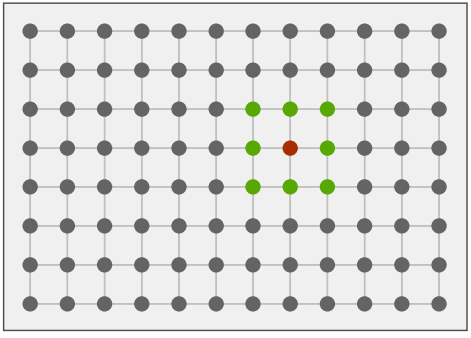
Motivation

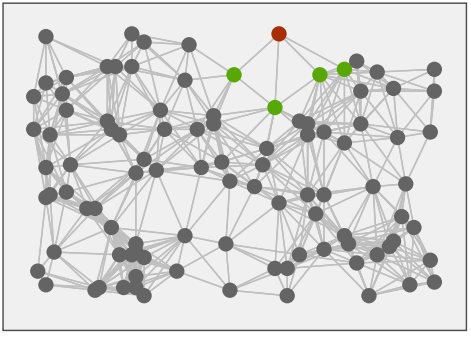




- We know how to build neural nets over larger **grid** structured domains (CNNs)
- How can we build neural nets for general **graph** structured inputs? (GNNs)
- Goal: We want to exploit the **local structure** of the graph

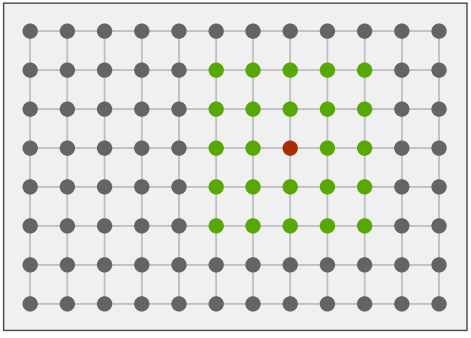
Motivation

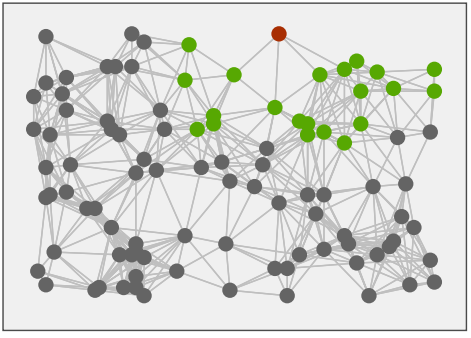




- Graphs are descriptors of the **signal structure** with signals at nodes and edges expressing similarity between signal components
- Convolutions on graphs are polynomials conditioned on the graph structure

Motivation



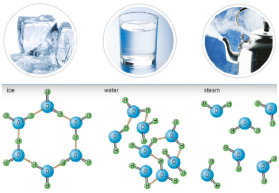


- Graphs are descriptors of the **signal structure** with signals at nodes and edges expressing similarity between signal components
- Convolutions on graphs are polynomials conditioned on the graph structure

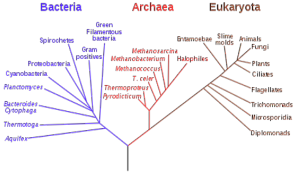
Applications

Applications

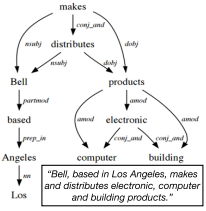
Molecules



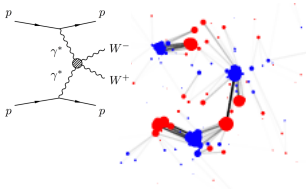
Biological species



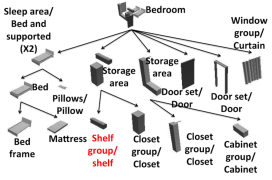
Natural language



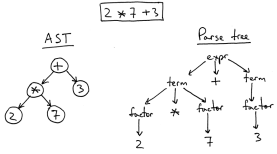
Sub-atomic particles



Everyday scenes

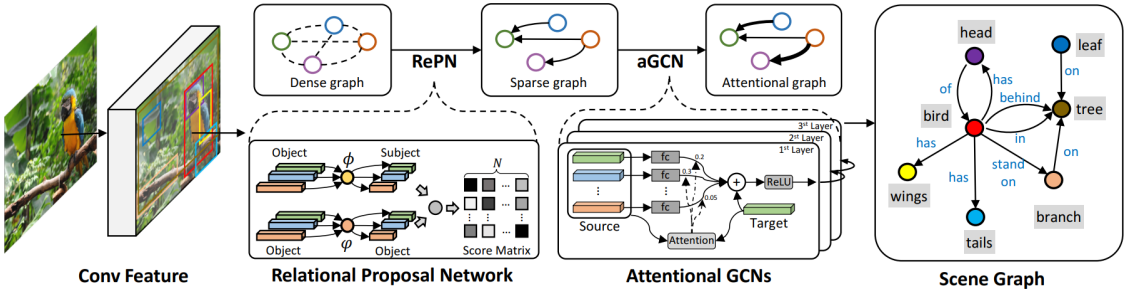


Code



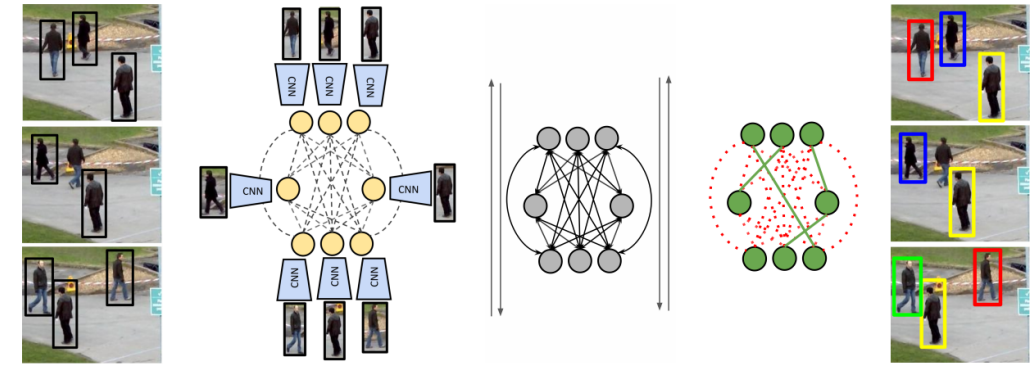
[Slide Credit: Peter Battaglia]

Scene Graph Generation



- Goal: Infer graph that captures semantic relationship of objects in image

Multiple Object Tracking



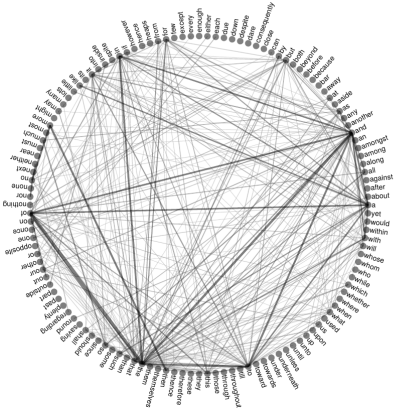
(a) Input (b) Graph Construction + Feature Encoding (c) Neural Message Passing (d) Edge Classification (e) Output

- Goal: Infer multiple object trajectories over time

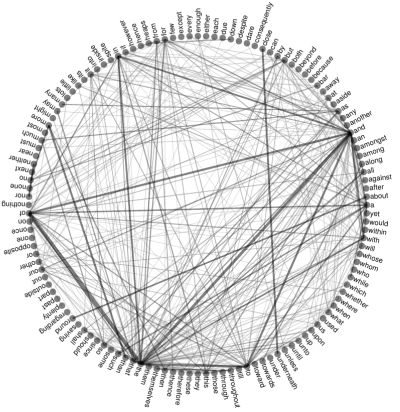
Brasó and Leal-Taixé: Learning a Neural Solver for Multiple Object Tracking. CVPR, 2020.

Authorship Attribution

William Shakespeare



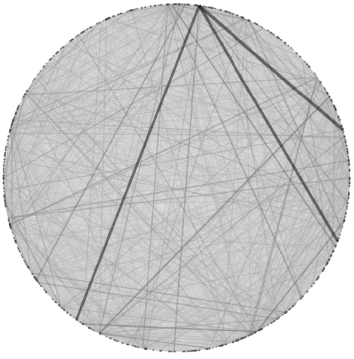
Christopher Marlowe



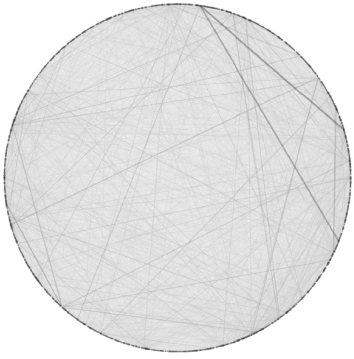
- Edges represent how often different function words appear close to each other

Recommendation System

Variation Diagram for Original (sampled) ratings



Variation Diagram for Reconstructed (predicted) ratings



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

Learning Molecular Fingerprints

Fragments most activated by toxicity feature on NR-AHR dataset

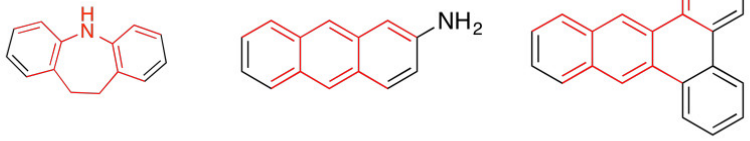


Figure 5: Visualizing fingerprints optimized for predicting toxicity. Shown here are representative samples of molecular fragments (highlighted in red) which most activate the feature most predictive of toxicity. *Top row:* the most predictive feature identifies groups containing a sulphur atom attached to an aromatic ring. *Bottom row:* the most predictive feature identifies fused aromatic rings, also known as polycyclic aromatic hydrocarbons, a well-known carcinogen.

Protein Interface Prediction

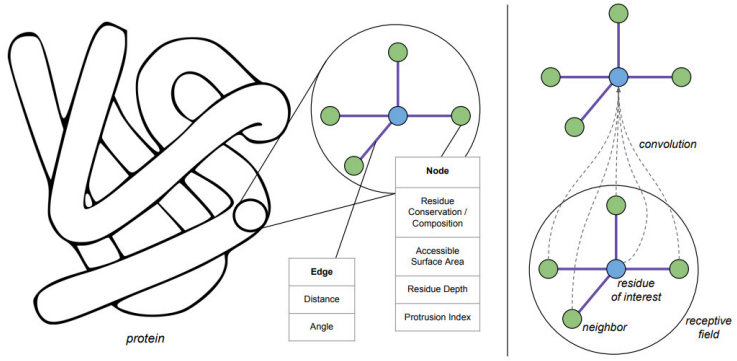
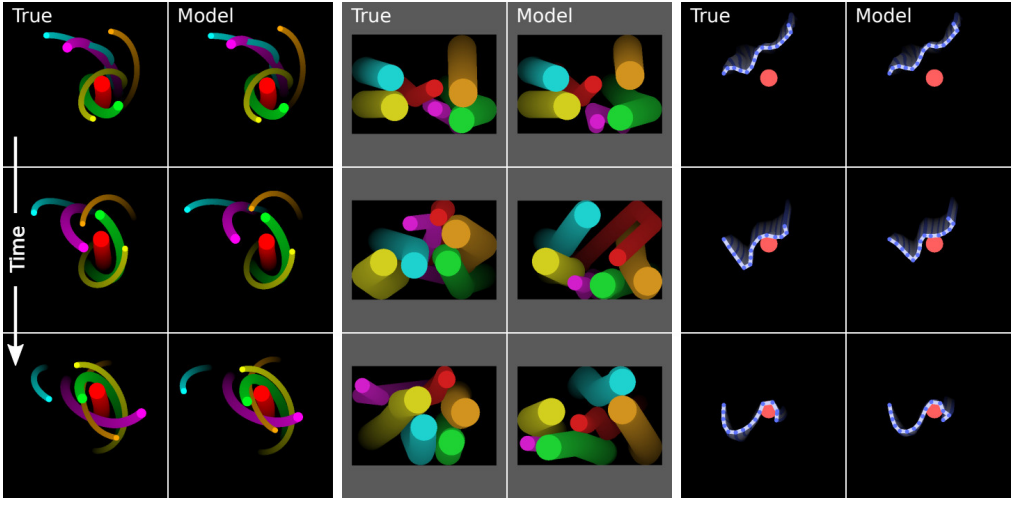


Figure 1: Graph convolution on protein structures. Left: Each residue in a protein is a node in a graph where the neighborhood of a node is the set of neighboring nodes in the protein structure; each node has features computed from its amino acid sequence and structure, and edges have features describing the relative distance and angle between residues. Right: Schematic description of the convolution operator which has as its receptive field a set of neighboring residues, and produces an activation which is associated with the center residue.

Fout, Byrd, Shariat and Ben-Hur: Protein Interface Prediction using Graph Convolutional Networks. NIPS, 2017.

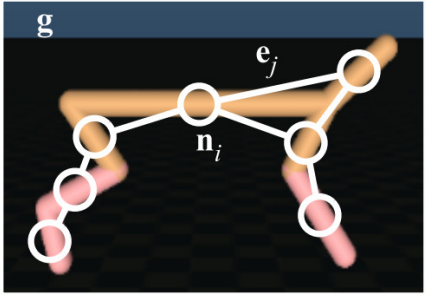
Interaction Networks



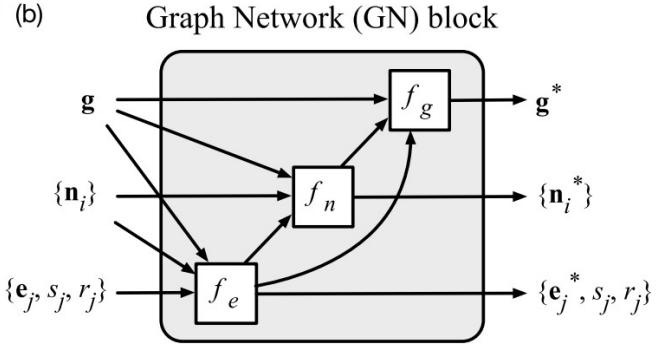
Battaglia et al.: Interaction Networks for Learning about Objects, Relations and Physics. NIPS, 2016

Learnable Physics Engines for Control

(a)



(b)



Sanchez-Gonzalez et al.: Graph Networks as Learnable Physics Engines for Inference and Control. ICML, 2018.

Decentralized Control of Robot Swarms

Tolstaya, Gama, Paulos, Pappas, Kumar and Ribeiro: Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks. CoRL, 2019.

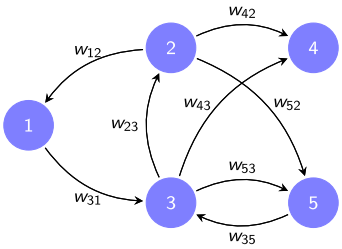
Remark

- This lecture is based on the class “Graph Neural Networks” of Alejandro Ribeiro at the University of Pennsylvania
- In particular, it covers lectures 3 and 4 of that class (partially)
- I highly recommend to have a look at this lecture if you are interested in diving more deeply into the topic of graph neural networks
- `https://gnn.seas.upenn.edu/`

10.2
Graph Convolution Filters

Graphs

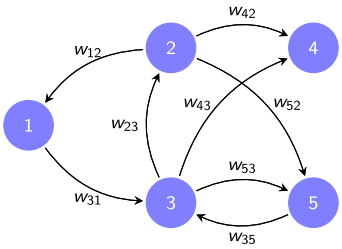
Directed Graphs



A **directed graph** is a triplet $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ with vertices $\mathcal{V}$, edges $\mathcal{E}$ and weights $\mathcal{W}$

- Vertices or nodes are defined as a set of N labels: $\mathcal{V} = \{1, \ldots, N\}$
- Edges are ordered pairs of lables (i, j). We interpret $(i, j) \in \mathcal{E}$ as i influenced by j
- Weights $w_{ij} \in \mathbb{R}$ are numbers associated to edges (i, j) that determine the strength of the influence that node j has on node i

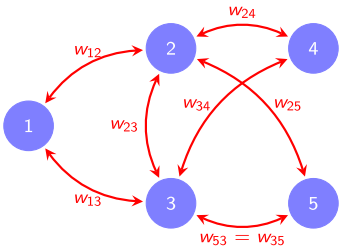
Directed Graphs



A **directed graph** is a triplet $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ with vertices $\mathcal{V}$, edges $\mathcal{E}$ and weights $\mathcal{W}$

- Edge (i, j) is represented by an arrow from node j to the influenced node i
- Edge (i, j) is different from edge (j, i): we can have $(i, j) \in \mathcal{E}$ and $(j, i) \notin \mathcal{E}$
- If $\{(i, j), (j, i)\} \subseteq \mathcal{E}$, their weights can be different: $w_{ij} \neq w_{ji}$

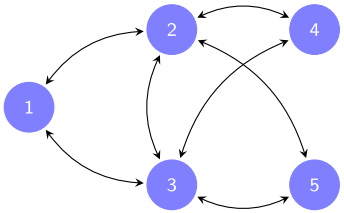
Symmetric Graphs



A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is **symmetric** if the edge set and weights are symmetric

- The edge set is symmetric if $(i, j) \in \mathcal{E}$ implies that $(j, i) \in \mathcal{E}$
- The weights are symmetric if $w_{ij} = w_{ji}$ for all $(i, j) \in \mathcal{E}$

Unweighted Graphs

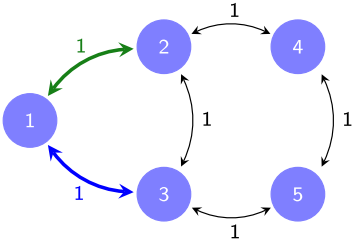


A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is **unweighted** if it doesn't have weights

- Equivalently, we can say that all weights are one $\Rightarrow w_{ij} = 1$ for all $(i, j) \in \mathcal{E}$
- Unweighted graphs can be directed or symmetric
- But most graphs we encounter in practice are weighted

Graph Shift Operators

Adjacency Matrices



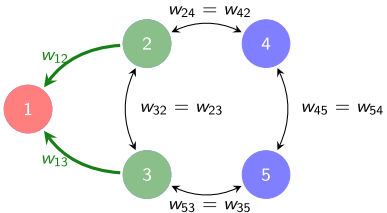
$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

- The **adjacency matrix** of graph $\mathcal{G}$ is the sparse matrix $\mathbf{A}$ with nonzero entries:

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

- If the graph is symmetric, the adjacency matrix is symmetric $\Rightarrow \mathbf{A} = \mathbf{A}^\top$

Neighborhoods and Degree



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

Node 1 degree $\Rightarrow n(1) = w_{12} + w_{13}$

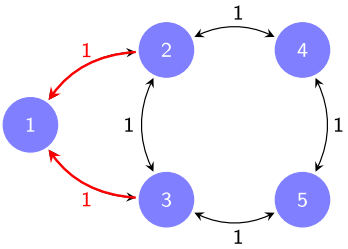
- The **neighborhood** $\mathcal{N}(i)$ of node i is the set of nodes that influence node i:

$$\mathcal{N}(i) = \{j | (i, j) \in \mathcal{E}\}$$

- The **degree** d_i of node i is the sum of weights of its incident edges:

$$d_i = \sum_{j \in \mathcal{N}(i)} w_{ij}$$

Degree Matrix



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

- The **degree matrix** $\mathbf{D}$ is a diagonal matrix of degrees: $D_{ii} = d_i$
- We can write $\mathbf{D}$ in terms of the adjacency matrix: $\mathbf{D} = \mathrm{diag}(\mathbf{A1})$

Laplacian Matrix

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

- The **Laplacian** matrix $\mathbf{L}$ of a graph with adjacency matrix $\mathbf{A}$ is defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

- It can also be written explicitly in terms of graph weights
  - Off diagonal entries: $L_{ij} = -A_{ij} = -w_{ij}$
  - Diagonal entries: $L_{ii} = d_i = \sum_{j \in \mathcal{N}(i)} w_{ij}$

Normalized Adjacency and Laplacian Matrix

- The **normalized adjacency matrix** expresses weights relative to node degrees

$$\bar{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \quad \Rightarrow \quad \bar{A}_{ij} = \frac{w_{ij}}{\sqrt{d_i d_j}}$$

- The **normalized Laplacian matrix** is similarly defined as

$$\bar{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \bar{\mathbf{A}}$$

- Normalized operators are more **homogeneous**

Graph Shift Operator

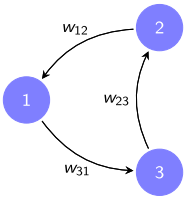
- The **Graph Shift Operator** $\mathbf{S}$ represents any of the graph matrix representations

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 is symmetric: $\mathbf{S} = \mathbf{S}^\top$
- The specific choice of $\mathbf{S}$ matters in practice but the analysis holds for any choice

Graph Signals

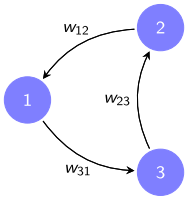
Graph Signal Diffusion



$$\underbrace{\begin{pmatrix}5\\0\\10\end{pmatrix}}_{=\mathbf{y}} = \underbrace{\begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix}}_{=\mathbf{S}} \underbrace{\begin{pmatrix}10\\5\\0\end{pmatrix}}_{=\mathbf{x}} \qquad \underbrace{\begin{pmatrix}0\\10\\5\end{pmatrix}}_{=\mathbf{y}} = \underbrace{\begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix}}_{=\mathbf{S}} \underbrace{\begin{pmatrix}5\\0\\10\end{pmatrix}}_{=\mathbf{x}}$$

- Consider a graph $\mathcal{G}$ with N nodes and shift operator $\mathbf{S}$
- A **graph signal** is a vector $\mathbf{x} \in \mathbb{R}^N$ that assigns a value $x_i \in \mathbb{R}$ to every node i
- $\mathbf{S}$ encodes expected proximity or similarity between components of $\mathbf{x}$

Graph Signal Diffusion



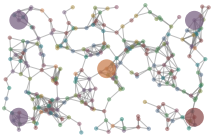
$$\underbrace{\begin{pmatrix}5\\0\\10\end{pmatrix}}_{=\mathbf{y}} = \underbrace{\begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix}}_{=\mathbf{S}} \underbrace{\begin{pmatrix}10\\5\\0\end{pmatrix}}_{=\mathbf{x}} \qquad \underbrace{\begin{pmatrix}0\\10\\5\end{pmatrix}}_{=\mathbf{y}} = \underbrace{\begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix}}_{=\mathbf{S}} \underbrace{\begin{pmatrix}5\\0\\10\end{pmatrix}}_{=\mathbf{x}}$$

- Multiplication by the graph shift operator yields a **diffused signal** over the graph:

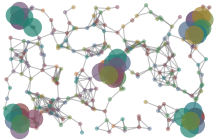
$$\mathbf{y} = \mathbf{S}\,\mathbf{x} \quad \text{or} \quad y_i = \sum_j w_{ij} x_j$$

- Implements a local operation where values are mixed with neighboring values

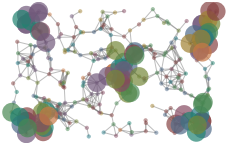
The Diffusion Sequence



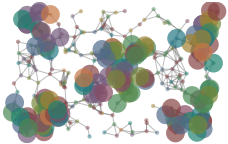
$\mathbf{S}^0\mathbf{x} = \mathbf{x}$



$\mathbf{S}^1\mathbf{x}$



$\mathbf{S}^2\mathbf{x}$



$\mathbf{S}^3\mathbf{x}$

- We can apply $\mathbf{S}$ multiple times to obtain a **diffusion sequence**

$$\mathbf{x}_{k+1} = \mathbf{S}\,\mathbf{x}_k \quad \text{with} \quad \mathbf{x}_0 = \mathbf{x}$$

- Equivalently, we can write this diffusion process as a **power sequence**

$$\mathbf{x}_k = \mathbf{S}^k\,\mathbf{x}$$

Graph Convolution Filters

Graph Convolution Filters

- Given $\mathbf{S}$ and filter coefficients h_k, a **graph convolution filter** is a polynomial 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 $\mathbf{y} = \mathbf{h} \star_{\mathbf{S}} \mathbf{x}$ is the **graph convolution** of filter $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$ with signal $\mathbf{x}$

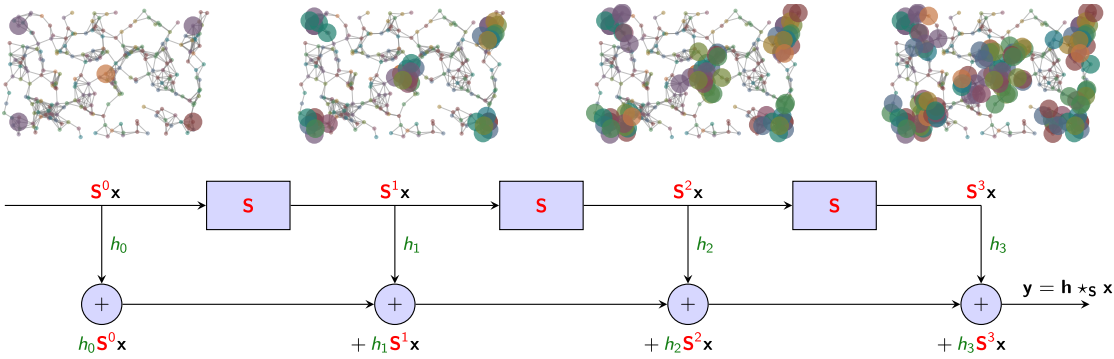
From Local to Global Information

- Graph convolutions **aggregate** information from local to global neighborhoods
- Consider a signal $\mathbf{x}$ supported on $\mathcal{G}$ with shift operator $\mathbf{S}$ and filter $\mathbf{h} = \{h_k\}_{k=0}^{K-1}$
- The output of the graph convolution with this **finite filter** is given by

$$\mathbf{y} = \mathbf{h} \star_{\mathbf{S}} \mathbf{x} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + \cdots + h_{K-1} \mathbf{S}^{K-1} \mathbf{x} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$$

- We see that a graph convolution is a **weighted linear combination** of the elements of the diffusion sequence $\mathbf{S}^k \mathbf{x}$ which we have discussed before

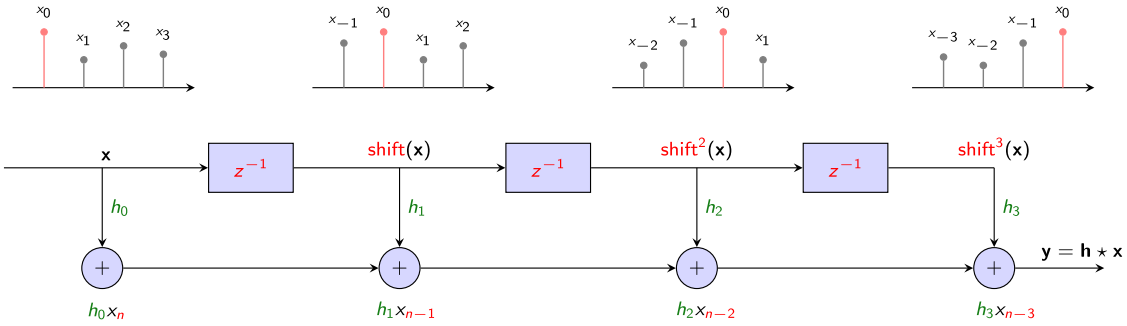
Graph Convolutions as Diffusion Operators



- Graph convolutions iteratively apply a shift, a weighting and a summation

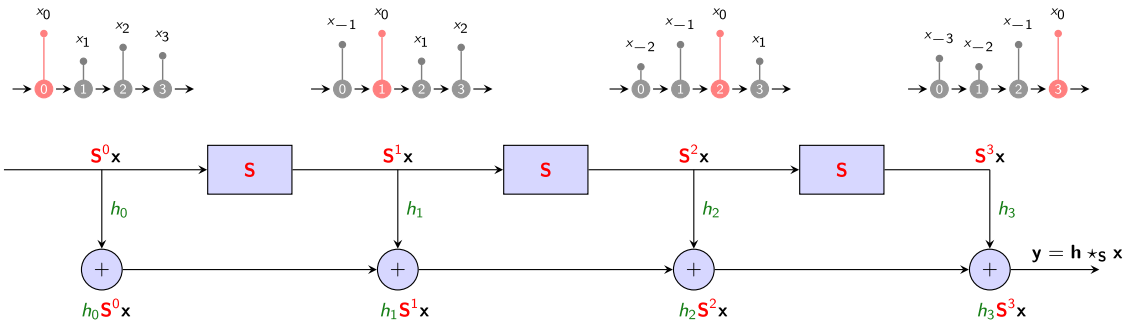
Time Convolutions as Graph Convolutions

Time Convolutions as Graph Convolutions



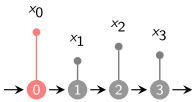
- **Time convolution** is a linear combination of shifted inputs: $y_n = \sum_{k=0}^{K-1} h_k \, x_{n-k}$

Time Convolutions as Graph Convolutions

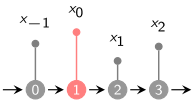


- Can we express time convolution via a graph convolution? On which graph?

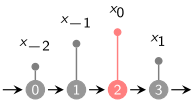
Time Convolutions as Graph Convolutions



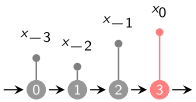
$\mathbf{x}$



$\mathbf{Sx}$



$\mathbf{S^2x}$

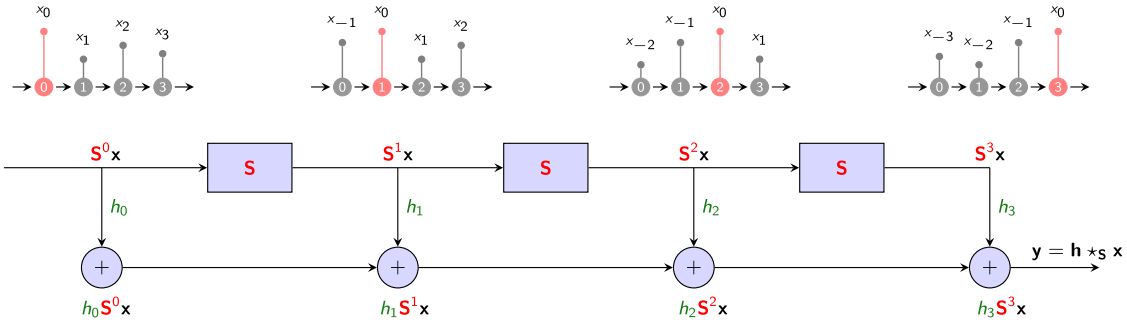


$\mathbf{S^3x}$

$$\underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}}_{=\mathbf{S}} \underbrace{\begin{pmatrix} x_{-1} \\ x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{=\mathbf{x}} = \underbrace{\begin{pmatrix} 0 \\ x_{-1} \\ x_0 \\ x_1 \\ x_2 \end{pmatrix}}_{=\mathbf{Sx}}$$

- Time convolution realized by multiplication with **adjacency matrix of line graph**

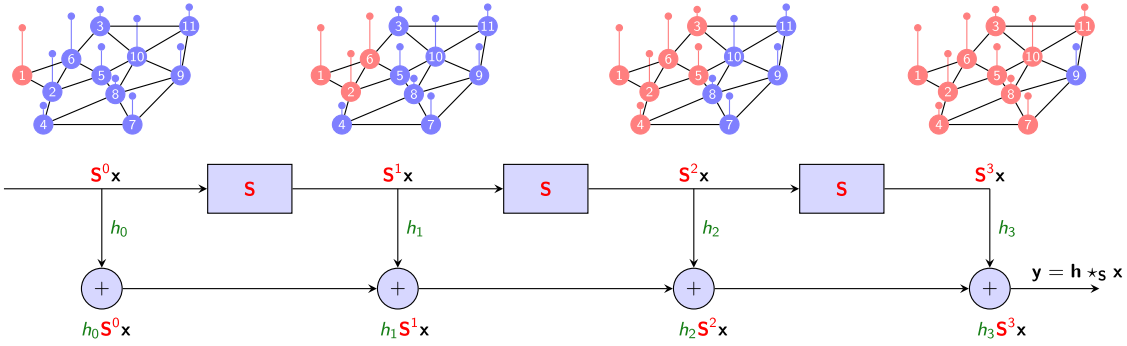
Time Convolutions as Graph Convolutions



- Time convolution is a **polynomial on the adjacency matrix of the line graph:**

$$\mathbf{y} = \sum_{k=0}^{K-1} h_k \, \mathbf{S}^k \, \mathbf{x}$$

Time Convolutions as Graph Convolutions



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

Graph Fourier Transform

Graph Fourier Transform

- Tool for analyzing graph information processing system
- Equivalent of standard Fourier Transform (FT) for graphs
- Given an eigendecomposition of the graph shift operator $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^*$, the **Graph Fourier Transform (GFT)** of graph signal is given by

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

- The GFT is thus a projection onto the eigenspace of the graph shift operator $\mathbf{S}$
- Graph convolutions are pointwise in the GFT domain: $\tilde{y}_i = \sum_{k=0}^{\infty} h_k \, \lambda_i^k \, \tilde{x}_i$
- Further reading: `https://gnn.seas.upenn.edu/` lectures 3 and 4

10.3
Graph Convolution Networks

Learning Graph Filters

Learning Graph Filters

- Let $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{S}_i)\}$ be a **dataset** of input/output signals on graphs
- Let $f_{\mathbf{h}}(\mathbf{x}, \mathbf{S})$ denote a **graph filter:**

$$\hat{\mathbf{y}} = f_{\mathbf{h}}(\mathbf{x}, \mathbf{S}) = \sum_{k=0}^{K-1} h_k \, \mathbf{S}^k \, \mathbf{x}$$

- During training, we seek to minimize a **loss function** $\mathcal{L}$:

$$\mathbf{h}^* = \underset{\mathbf{h}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y},\mathbf{S}) \in \mathcal{D}} \mathcal{L}\left(f_{\mathbf{h}}(\mathbf{x}, \mathbf{S}), \mathbf{y}\right)$$

- The optimization is over filter coefficients
- Inputs, outputs and the graph are given

Readout Layer

- When the output $\mathbf{y}^M$ is not a graph signal ($M \neq N$), we add a **readout layer**

$$\hat{\mathbf{y}} = f_{\mathbf{h}}(\mathbf{x}, \mathbf{S}) = \mathbf{R} \sum_{k=0}^{K-1} h_k \, \mathbf{S}^k \, \mathbf{x}$$

  with $\mathbf{R} \in \mathbb{R}^{M \times N}$ for matching dimensions.

- Typically, $\mathbf{R}$ is not learned but a design choice of the algorithm.
- Example for readout layers include:
  - Read out node $i \Rightarrow \mathbf{R} = \mathbf{e}_i^\top$
  - Read out summation $\Rightarrow \mathbf{R} = \mathbf{1}^\top$

Graph Convolution Networks

Kumar Bipin

Graph Perceptrons

- Graph filters have limited expressive power as they can only learn linear maps
- A first approach to realize non-linear mappings is the **graph perceptron**

$$f_{\mathbf{h}}(\mathbf{x}, \mathbf{S}) = g\left(\sum_{k=0}^{K-1} h_k \, \mathbf{S}^k \, \mathbf{x}\right)$$

  where $g(\cdot)$ is a point-wise non-linearity, such as a sigmoid, tanh or ReLU
- A perceptron allows for learning non-linear mapping $\Rightarrow$ larger function class

Graph Convolution Networks

- To define a **GCN**, we recursively compose several graph perceptrons in layers

$$\mathbf{x}_{\ell} = g\left(\sum_{k=0}^{K-1} h_{\ell k}\, \mathbf{S}^{k}\, \mathbf{x}_{\ell-1}\right)$$

  where we have assumed the input to the first layer to be $\mathbf{x}_0 = \mathbf{x}$

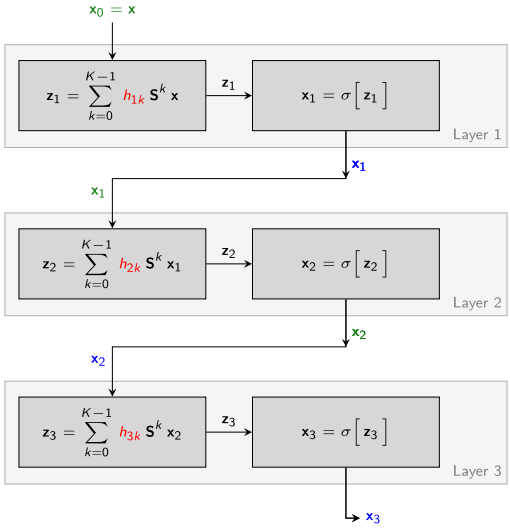
- For a GCN with L layers we write

$$\hat{\mathbf{y}} = f_{\mathcal{H}}(\mathbf{x}, \mathbf{S}) = \mathbf{x}_L$$

  where the filter of each layer $\mathcal{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_L\}$ are the trainable parameters

- The graph shift operator $\mathbf{S}$ is given as prior information

Block Diagram



Learning Graph Convolution Networks

- Let $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{S}_i)\}$ be a dataset of input/output signals on graphs
- Let $f_{\mathcal{H}}(\mathbf{x}, \mathbf{S})$ denote a GCN with filter set $\mathcal{H}$
- During training, we seek to minimize a **loss function $\mathcal{L}$:**

$$\mathcal{H}^* = \underset{\mathcal{H}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y},\mathbf{S})\in\mathcal{D}} \mathcal{L}\left(f_{\mathcal{H}}(\mathbf{x}, \mathbf{S}), \mathbf{y}\right)$$

- The optimization is over all filter coefficients of all layers $\mathcal{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_L\}$
- The graph shift operator $\mathbf{S}$ is given as prior information
- The dataset $\mathcal{D}$ may comprise multiple different graph instances
- Deeper GCNs work better due to the layered non-linearities (as for CNNs)

Comparison to Multi-Layer Perceptron

Multi-Layer Perceptron

$$\mathbf{x}_\ell = g\left(\mathbf{W}_\ell\, \mathbf{x}_{\ell-1}\right)$$

Graph Convolution Network

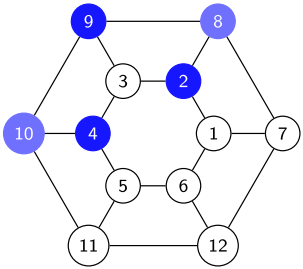
$$\mathbf{x}_\ell = g\left(\sum_{k=0}^{K-1} h_{\ell k}\, \mathbf{S}^k\, \mathbf{x}_{\ell-1}\right)$$

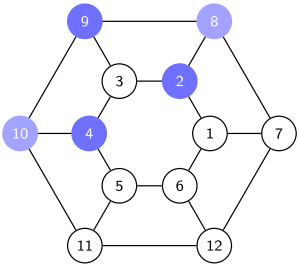
- The trainable parameters are highlighted in red
- **GCNs are a special case of MLPs** and thus attain higher training cost

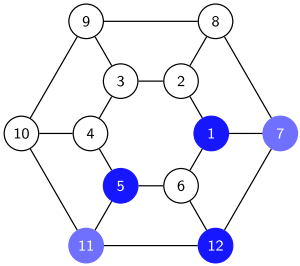
$$\min_{\mathcal{W}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \mathcal{L}\left(f_{\mathcal{W}}(\mathbf{x}),\mathbf{y}\right) \leq \min_{\mathcal{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \mathcal{L}\left(f_{\mathcal{H}}(\mathbf{x},\mathbf{S}),\mathbf{y}\right)$$

- However, they can handle **different graphs** ($\mathbf{S}$ must be constant for MLPs)
- They also **generalize better** to unseen signals as they are able to exploit internal symmetries of graph signals encoded in the graph shift operator $\mathbf{S}$

Generalization Properties





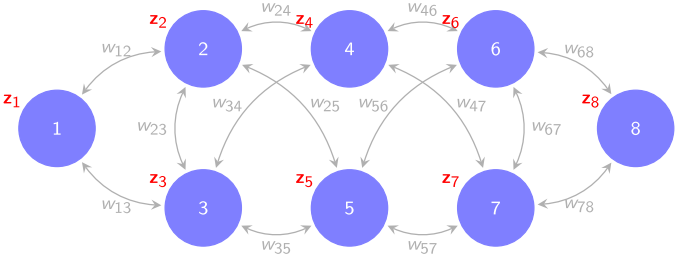


- Assume the training examples are similar to the signal shown on the left
- A **MLP** learns to predict the middle signal but not the right one
- The **GCN** will succeed as it knows the underlying structure of the graph

Multiple-Input-Multiple-Output GCNs

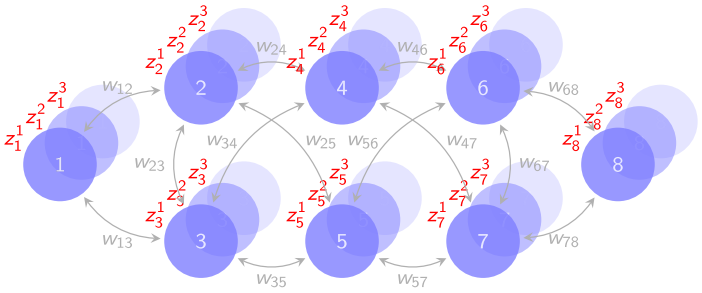
Filter Banks

- **Filter banks** output a graph signal $\mathbf{Z} = (\mathbf{z}^1, \ldots, \mathbf{z}^F)$ in matrix form
- The matrix graph signals $\mathbf{Z} = (\mathbf{z}^1, \ldots, \mathbf{z}^F)$ represents F features per node



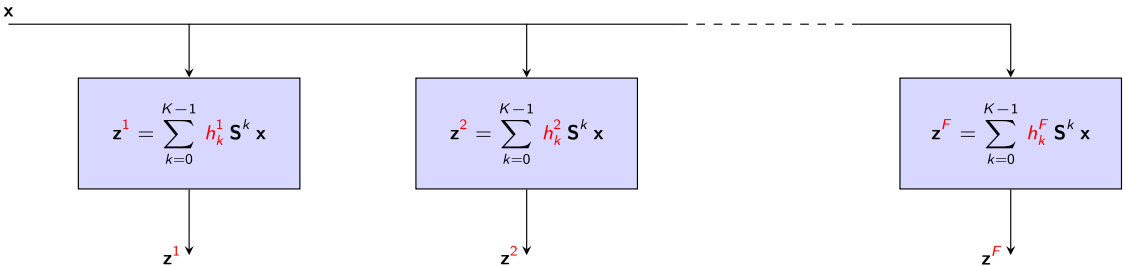
Filter Banks

- **Filter banks** output a graph signal $\mathbf{Z} = (\mathbf{z}^1, \ldots, \mathbf{z}^F)$ in matrix form
- The matrix graph signals $\mathbf{Z} = (\mathbf{z}^1, \ldots, \mathbf{z}^F)$ represents F features per node



Filter Banks

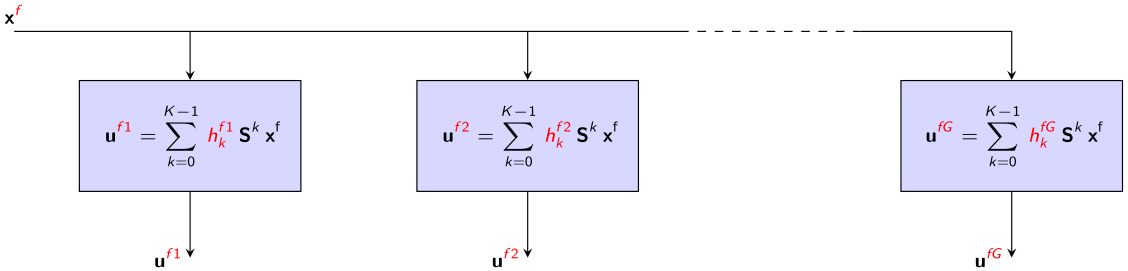
- **Filter banks** output a graph signal $\mathbf{Z} = (\mathbf{z}^1, \ldots, \mathbf{z}^F)$ in matrix form
- The matrix graph signals $\mathbf{Z} = (\mathbf{z}^1, \ldots, \mathbf{z}^F)$ represents F features per node



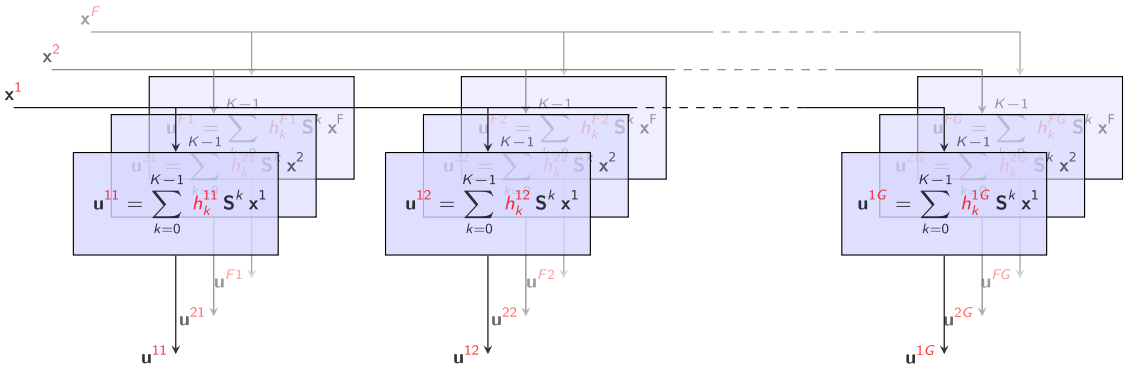
Filter Banks

- Each of the F features $\mathbf{x}^f$ is processed with G filters with coefficients h_k^{fg}:

$$\mathbf{u}^{fg} = \sum_{k=0}^{K-1} h_k^{fg} \mathbf{S}^k \mathbf{x}^f$$



Multiple-Input-Multiple-Output Graph Filters



- The MIMO Graph Filter thus generates an output with $F \times G$ features per node
- Summing reduces this to G features: $\mathbf{z}^g = \sum_{f=1}^{F} \mathbf{u}^{fg} = \sum_{f=1}^{F} \sum_{k=0}^{K-1} h_k^{fg} \mathbf{S}^k \mathbf{x}^f$

Multiple-Input-Multiple-Output Graph Filters

- Using matrix notation, the **MIMO Graph Filter** operation can be written as

$$\mathbf{z}^g = \sum_{f=1}^{F} \sum_{k=0}^{K-1} h_k^{fg} \, \mathbf{S}^k \, \mathbf{x}^f \quad \Leftrightarrow \quad \mathbf{Z} = \sum_{k=0}^{K-1} \mathbf{S}^k \, \mathbf{X} \, \mathbf{H}_k$$

with

$$\underbrace{\begin{pmatrix} \mathbf{z}^1 & \cdots & \mathbf{z}^G \end{pmatrix}}_{= \mathbf{Z} \in \mathbb{R}^{N \times G}} = \sum_{k=0}^{K-1} \underbrace{\mathbf{S}^k}_{\in \mathbb{R}^{N \times N}} \underbrace{\begin{pmatrix} \mathbf{x}^1 & \cdots & \mathbf{x}^F \end{pmatrix}}_{= \mathbf{X} \in \mathbb{R}^{N \times F}} \underbrace{\begin{pmatrix} h_k^{11} & \cdots & h_k^{1G} \\ \vdots & & \vdots \\ h_k^{F1} & \cdots & h_k^{FG} \end{pmatrix}}_{= \mathbf{H}_k \in \mathbb{R}^{F \times G}}$$

Multiple-Input-Multiple-Output GCNs

- To define a **MIMO GCN,** we recursively compose L MIMO Filter layers

$$\mathbf{X}_{\ell} = g\left(\sum_{k=0}^{K-1} \mathbf{S}^{k}\, \mathbf{X}_{\ell-1}\, \mathbf{H}_{\ell k}\right)$$

  with $\mathbf{X}_0 = \mathbf{X}$ the input to the first layer

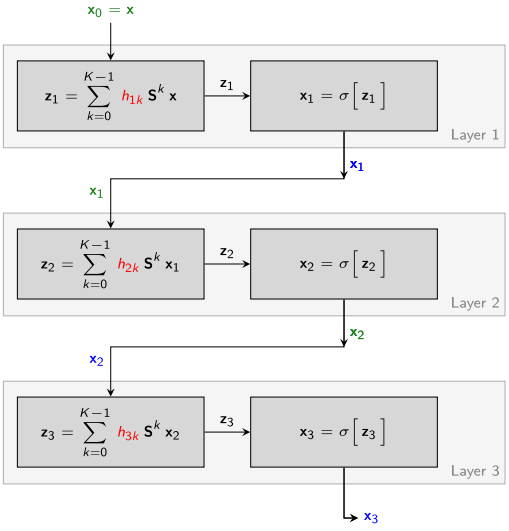
- For a MIMO GCN with L layers we write

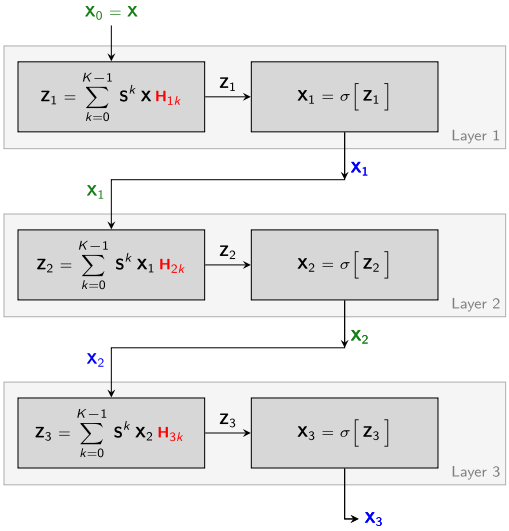
$$\hat{\mathbf{Y}} = f_{\mathcal{H}}(\mathbf{X}, \mathbf{S}) = \mathbf{X}_L$$

  with parameters $\mathcal{H} = \{\mathbf{H}_1, \ldots, \mathbf{H}_L\}$ where $\mathbf{H}_{\ell} = \{\mathbf{H}_{\ell 0}, \ldots, \mathbf{H}_{\ell, K-1}\}$

- The graph shift operator $\mathbf{S}$ is given as prior information

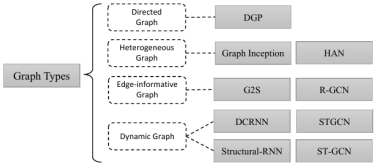
GCN vs. MIMO GCN



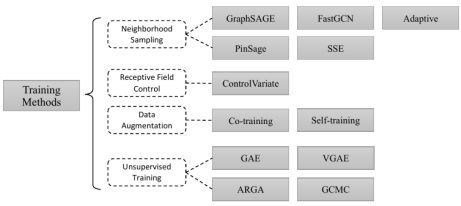


Concluding Remark

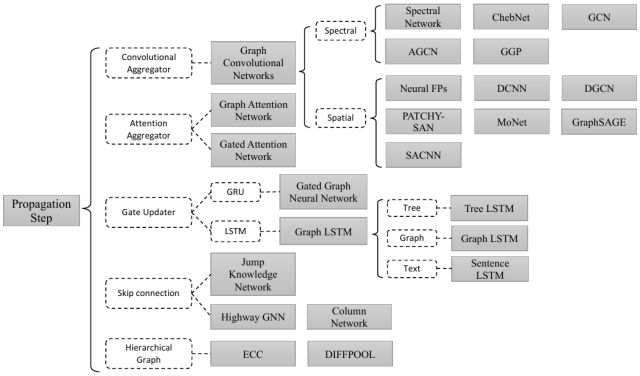
We just scratched the surface ..



(a) Graph Types



(b) Training Methods



(c) Propagation Steps

Zhou, Cui, Zhang, Yang, Liu and Sun: Graph Neural Networks: A Review of Methods and Applications. Arxiv, 2018.