Graph Neural Networks

Machine Learning on Graphs, Class 2

Fernando Gama

Electrical Engineering and Computer Sciences Department, University of California, Berkeley

Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay

February 2nd, 2021

Thanks: D. Owerko, E. Tolstaya, L. Ruiz, Q. Li, T.-K. Hu, A. Prorok, A. G. Marques, Z. Wang, E. Isufi, G. Leus, J. Bruna, S. Sojoudi and **A. Ribeiro**

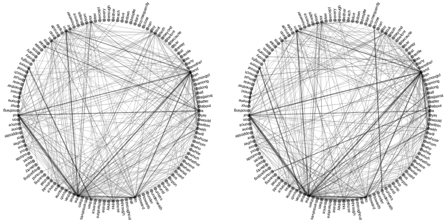
Berkeley
UNIVERSITY OF CALIFORNIA

Machine Learning on Graphs

F. Gama
fgama@berkeley.edu

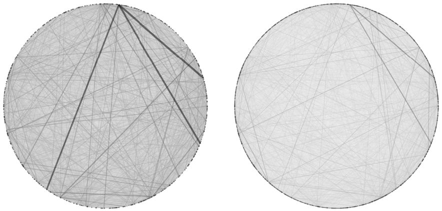
- Graphs are generic models of signal structure that can help to learn in several practical problems

Authorship Attribution



[Segarra et al, '16]

Recommendation Systems



[Ruiz et al '18]

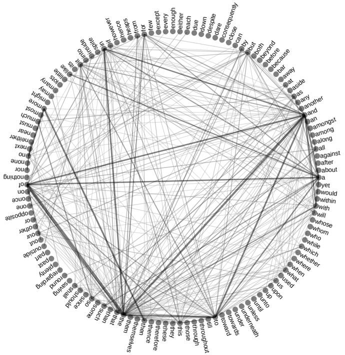
- In both cases there exists a graph that contains meaningful information about the problem to solve

Authorship Attribution with Word Adjacency Networks (WANs)

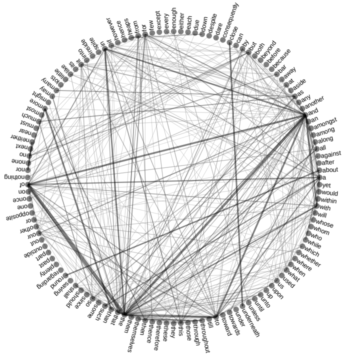
F. Gama
fgama@berkeley.edu

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

William Shakespeare



Christopher Marlowe



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

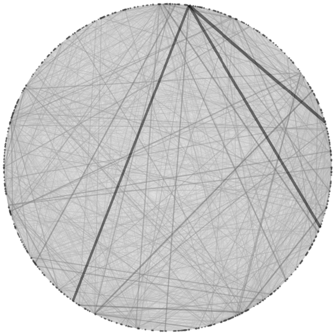
Segarra, Eisen, Egan, Ribeiro, "Attributing the Authorship of the Henry VI Plays by Word Adjacency", Shakespeare Quarterly 2016

Recommendation with Collaborative Filtering

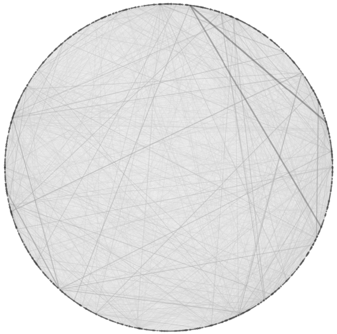
F. Gama
fgama@berkeley.edu

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

Original (sampled) ratings



Reconstructed (predicted) ratings



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

Ruiz, Gama, Marques, Ribeiro, "Invariance-Preserving Localized Activation Functions for Graph Neural Networks", TSP 2019.

Graphs in Physical Systems

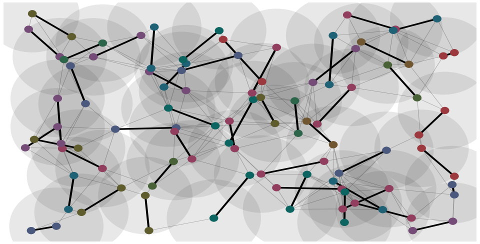
F. Gama
fgama@berkeley.edu

- Graphs are more than data structures $\Rightarrow$ They are models of the bulk of our physical infrastructure

Decentralized Control of Autonomous Systems

[Tolstaya et al, '19]

Wireless Communications Networks



Eisen and Ribeiro, '19

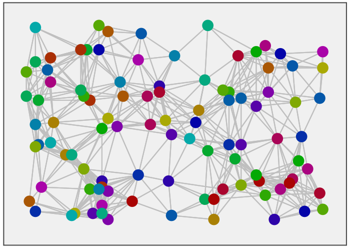
- The graph is the source of the problem $\Rightarrow$ Challenge is that goals are global but information is local

Neural Networks and Convolutional Neural Networks

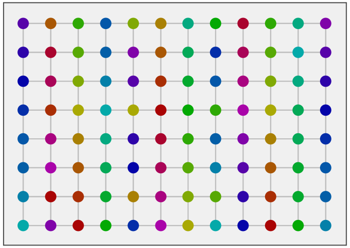
F. Gama
fgama@berkeley.edu

- This answers the *Why* $\Rightarrow$ What about the *How*?
- There is overwhelming empirical and theoretical justification to choose a neural network (NN)

Challenge is we want to run a NN over this



But we are good at running NNs over this



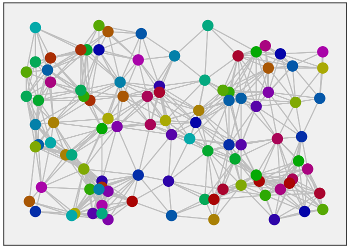
- Generic NNs do not scale to large dimensions $\Rightarrow$ Convolutional Neural Networks (CNNs) do

Convolutional Neural Networks and Graph Neural Networks

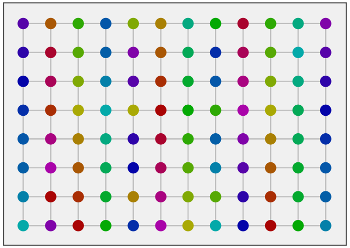
F. Gama
fgama@berkeley.edu

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

Process graphs with graph convolutional NNs



Process images with convolutional NNs



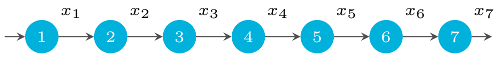
- Generalize convolutions to graphs and compose graph filter banks with pointwise nonlinearities
- Stack in layers to create a graph (convolutional) Neural Network (GNN)

Time and Space are Representable by Graphs

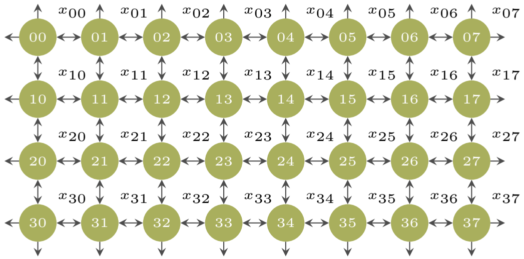
F. Gama
fgama@berkeley.edu

- How do we generalize convolutions in time and space to operate on graphs?
  - $\Rightarrow$ We can describe discrete time and space using graphs that support time or space signals

Description of time with a directed line graph



Description of images (space) with a grid graph


00 01 02 03 04 05 06 07
10 11 12 13 14 15 16 17
20 21 22 23 24 25 26 27
30 31 32 33 34 35 36 37
x_{00} x_{01} x_{02} x_{03} x_{04} x_{05} x_{06} x_{07}
x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
x_{20} x_{21} x_{22} x_{23} x_{24} x_{25} x_{26} x_{27}
x_{30} x_{31} x_{32} x_{33} x_{34} x_{35} x_{36} x_{37}

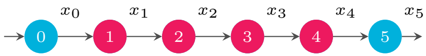
- Line graph represents adjacency of points in time. Grid graph represents adjacency of points in space

Convolutions in Time and Space

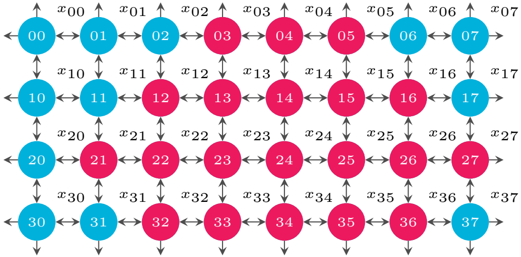
F. Gama
fgama@berkeley.edu

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph



Description of images (space) with a grid graph

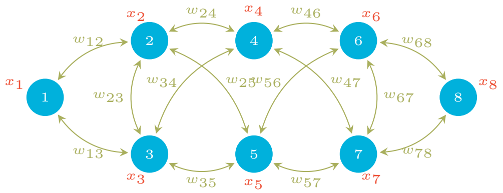

x_{00} x_{01} x_{02} x_{03} x_{04} x_{05} x_{06} x_{07}
00 01 02 03 04 05 06 07
x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
10 11 12 13 14 15 16 17
x_{20} x_{21} x_{22} x_{23} x_{24} x_{25} x_{26} x_{27}
20 21 22 23 24 25 26 27
x_{30} x_{31} x_{32} x_{33} x_{34} x_{35} x_{36} x_{37}
30 31 32 33 34 35 36 37

- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0\,\mathbf{S}^0\mathbf{x} + h_1\,\mathbf{S}^1\mathbf{x} + h_2\,\mathbf{S}^2\mathbf{x} + h_3\,\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\,\mathbf{S}^k\mathbf{x}$

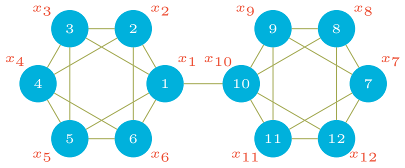
Arbitrary Graphs and Arbitrary Graph Signals

- Time and Space are pervasive and important, but still a (very) limited class of signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph



Another signal supported on another graph



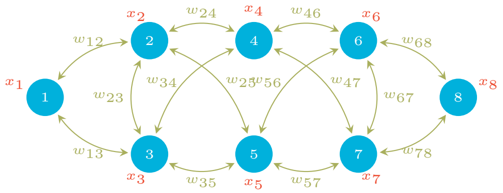
- Nodes are products. Signal values are product ratings. Edges are cosine similarities of past scores

Arbitrary Graphs and Arbitrary Graph Signals

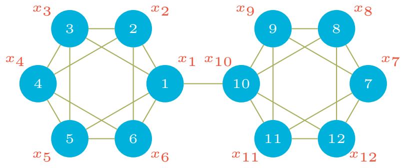
F. Gama
fgama@berkeley.edu

- Time and Space are pervasive and important, but still a (very) limited class of signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph



Another signal supported on another graph



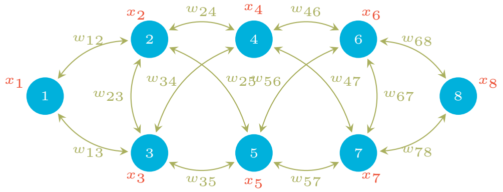
- Nodes are drones. Signal values are velocities. Edges are sensing and communication ranges

Arbitrary Graphs and Arbitrary Graph Signals

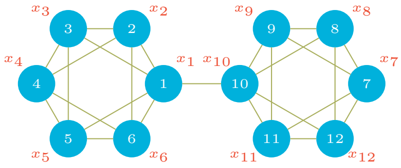
F. Gama
fgama@berkeley.edu

- Time and Space are pervasive and important, but still a (very) limited class of signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph



Another signal supported on another graph



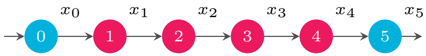
- Nodes are transceivers. Signal values are QoS requirements. Edges are wireless channels strength

Convolutions on Graphs

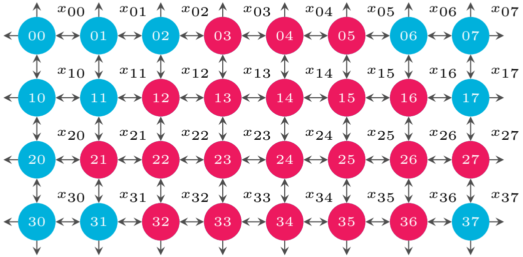
F. Gama
fgama@berkeley.edu

- We've already seen that convolutions in time and space are polynomials on adjacency matrices

Description of time with a directed line graph



Description of images (space) with a grid graph

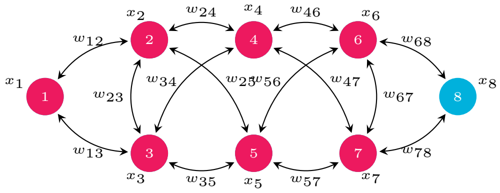

x_{00} x_{01} x_{02} x_{03} x_{04} x_{05} x_{06} x_{07}
00 01 02 03 04 05 06 07
x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
10 11 12 13 14 15 16 17
x_{20} x_{21} x_{22} x_{23} x_{24} x_{25} x_{26} x_{27}
20 21 22 23 24 25 26 27
x_{30} x_{31} x_{32} x_{33} x_{34} x_{35} x_{36} x_{37}
30 31 32 33 34 35 36 37

- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0\,\mathbf{S}^0\mathbf{x} + h_1\,\mathbf{S}^1\mathbf{x} + h_2\,\mathbf{S}^2\mathbf{x} + h_3\,\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\,\mathbf{S}^k\mathbf{x}$

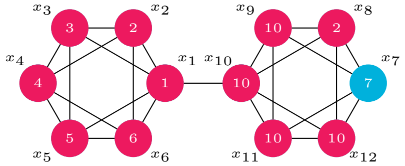
Convolutions on Graphs

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph

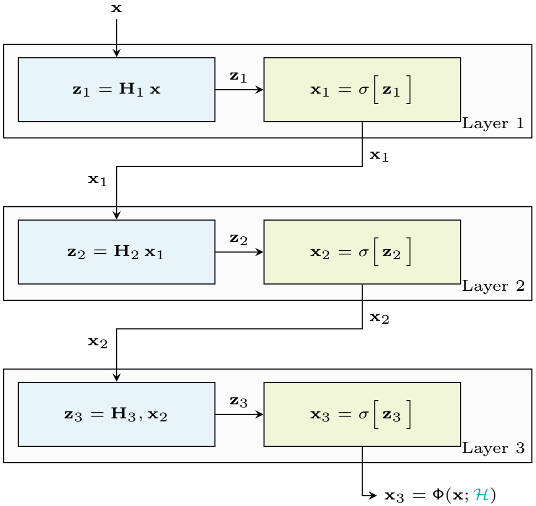


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0\,\mathbf{S}^0\mathbf{x} + h_1\,\mathbf{S}^1\mathbf{x} + h_2\,\mathbf{S}^2\mathbf{x} + h_3\,\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\,\mathbf{S}^k\mathbf{x}$
- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

F. Gama
fgama@berkeley.edu

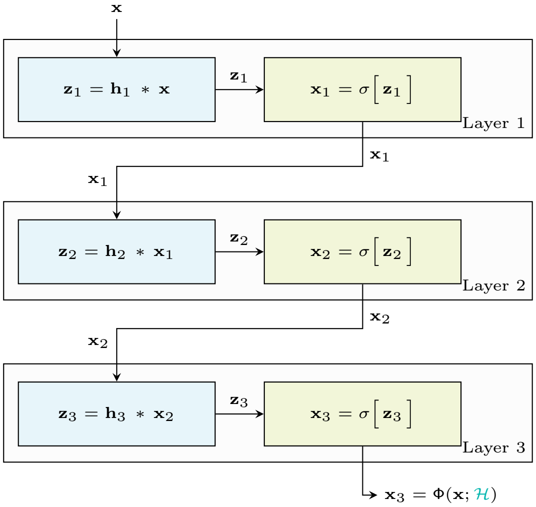
Neural Networks (NNs)

- 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 $\mathbf{x}$


$\mathbf{x}$
$\mathbf{z}_1 = \mathbf{H}_1\,\mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\left[\mathbf{z}_1\right]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \mathbf{H}_2\,\mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\left[\mathbf{z}_2\right]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \mathbf{H}_3, \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\left[\mathbf{z}_3\right]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathcal{H})$

Convolutional Neural Networks (CNNs)

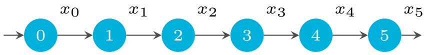
- A convolutional NN composes a cascade of layers
- Each of which are themselves compositions of convolutions with pointwise nonlinearities
- Scales well. The Deep Learning workhorse
- A CNNs are minor variation of convolutional filters
  - $\Rightarrow$ Just add nonlinearity and compose
  - $\Rightarrow$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


$\mathbf{x}$
$\mathbf{z}_1 = \mathbf{h}_1 * \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma[\mathbf{z}_1]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \mathbf{h}_2 * \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma[\mathbf{z}_2]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \mathbf{h}_3 * \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma[\mathbf{z}_3]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathcal{H})$

When we Think of Time Signals as Supported on a Line Graph

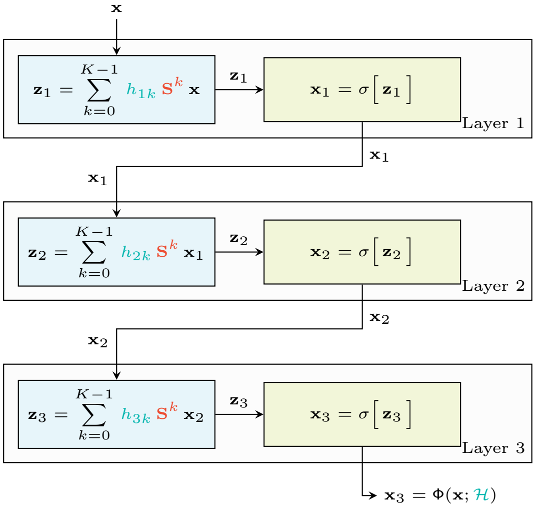
F. Gama
fgama@berkeley.edu

- 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

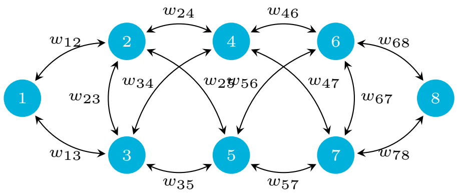
- But one that lends itself to generalization

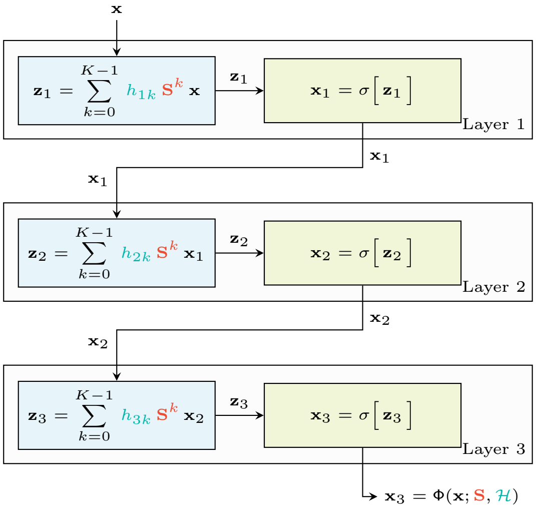

$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k}\,\mathbf{S}^k\,\mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\big[\mathbf{z}_1\big]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k}\,\mathbf{S}^k\,\mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\big[\mathbf{z}_2\big]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k}\,\mathbf{S}^k\,\mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\big[\mathbf{z}_3\big]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathcal{H})$

Graph Neural Networks (GNNs)

F. Gama
fgama@berkeley.edu

- The graph can be any arbitrary graph
- The polynomial on the matrix representation $\mathbf{S}$ becomes a graph convolutional filter

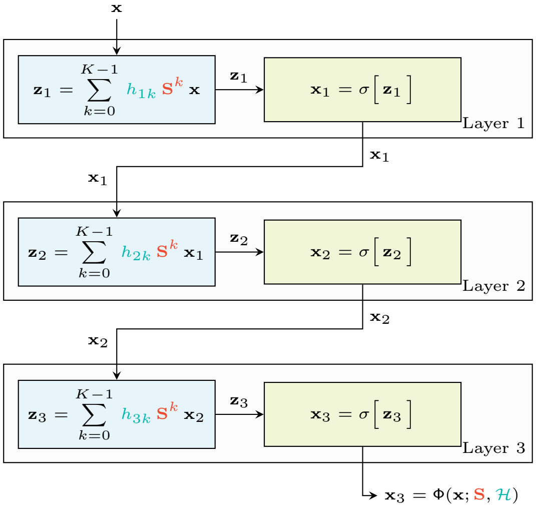



$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k} \mathbf{S}^k \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma[\mathbf{z}_1]$
Layer 1
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k} \mathbf{S}^k \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma[\mathbf{z}_2]$
Layer 2
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k} \mathbf{S}^k \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma[\mathbf{z}_3]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", TSP 2019

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


$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k} \mathbf{S}^k \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\left[\mathbf{z}_1\right]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k} \mathbf{S}^k \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\left[\mathbf{z}_2\right]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k} \mathbf{S}^k \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\left[\mathbf{z}_3\right]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

Graph Neural Networks (GNNs)

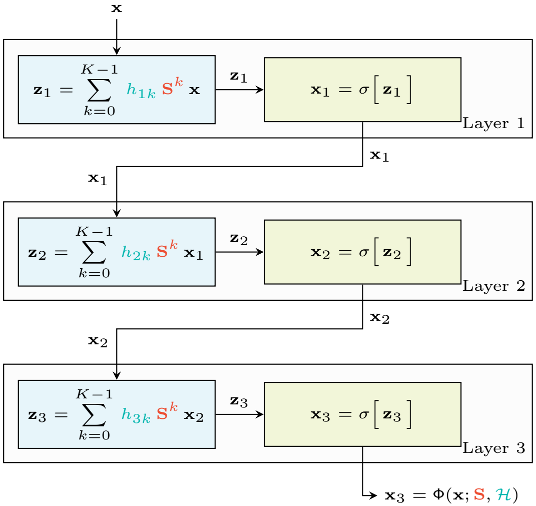
F. Gama
fgama@berkeley.edu

- There is growing evidence of scalability.

- A GNN is a minor variation of a graph filter

  $\Rightarrow$ Just add nonlinearity and compose

- Both are scalable because they leverage the signal structure codified by the graph


$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k} \mathbf{S}^k \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\left[\mathbf{z}_1\right]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k} \mathbf{S}^k \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\left[\mathbf{z}_2\right]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k} \mathbf{S}^k \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\left[\mathbf{z}_3\right]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

Outline

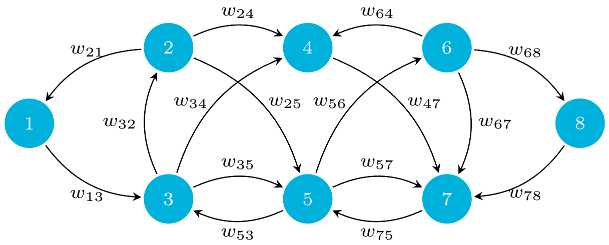
F. Gama
fgama@berkeley.edu

Graphs, Graph Signals and Convolutions

F. Gama
fgama@berkeley.edu

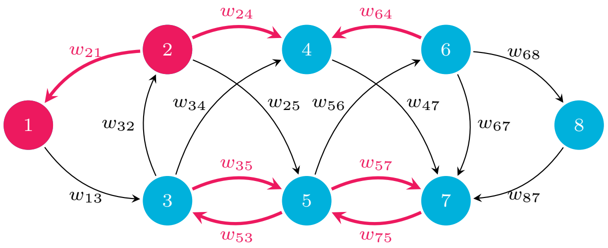
Nodes, Edges and Weights

- A graph is a triplet $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathsf{W})$ of vertices, edges, and weights
  - ⇒ Vertices or nodes are a set of N labels. Typical labels are $\mathcal{V} = \{1, \ldots, N\}$ or $\mathcal{V} = \{v_1, \ldots, v_N\}$
  - ⇒ Edges are ordered pairs of labels (i, j). We interpret $(i, j) \in \mathcal{E}$ as "i can influence j."
  - ⇒ Weights $w_{ij} \in \mathbb{R}$ are numbers associated to edges (i, j). "Strength of the influence of i on j."



Directed Graphs

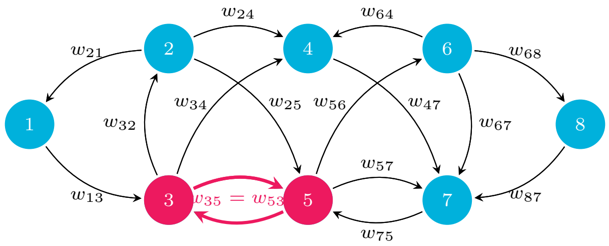
- Edge (i, j) is represented by an arrow pointing from i into j. Influence of node i on node j
- Edge (i, j) is different from edge (j, i) $\Rightarrow$ It is possible to have $(i, j) \in \mathcal{E}$ and $(j, i) \notin \mathcal{E}$
- If both edges are in the edge set, the weights can be different $\Rightarrow$ It is possible to have $w_{ij} \neq w_{ji}$



F. Gama
fgama@berkeley.edu

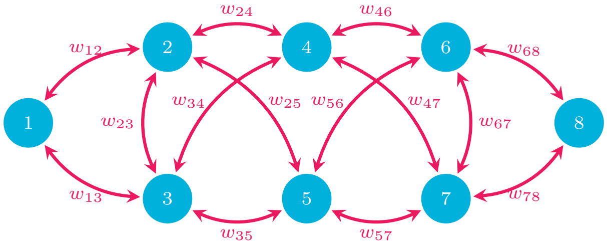
Undirected Graphs

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



Undirected Graphs

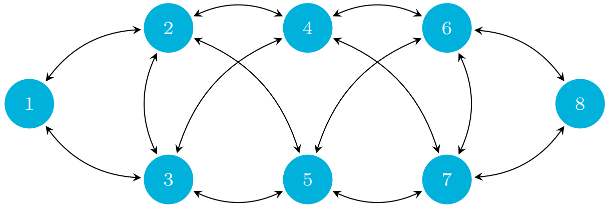
- A graph is undirected if both, the edge set and the weight are symmetric
  - $\Rightarrow$ Edges come in pairs $\Rightarrow$ We have $(i, j) \in \mathcal{E}$ if and only if $(j, i) \in \mathcal{E}$
  - $\Rightarrow$ Weights are symmetric $\Rightarrow$ We must have $w_{ij} = w_{ji}$ for all $(i, j) \in \mathcal{E}$



Unweighted Graphs

F. Gama
fgama@berkeley.edu

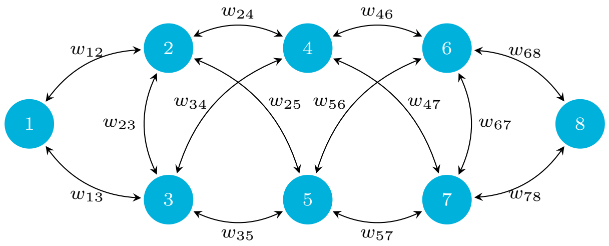
- A graph is unweighted if it doesn't have weights.
  - $\Rightarrow$ We, equivalently, have that all weights are units $\Rightarrow w_{ij} = 1$ for all $(i, j) \in \mathcal{E}$
- Unweighted graphs could be directed or undirected



Weighted Undirected Graphs

F. Gama
fgama@berkeley.edu

- Graphs can be directed or undirected. They can be weighted or unweighted
- For the most part we will work with graphs that are undirected and weighted



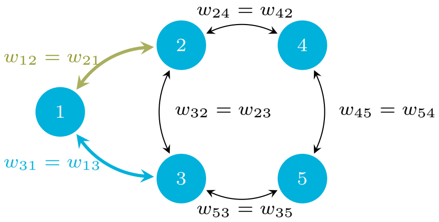
Adjacency Matrices

F. Gama
fgama@berkeley.edu

- The adjacency matrix of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathsf{W})$ is the sparse matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ with nonzero entries

$$[\mathbf{A}]_{ij} = w_{ji}, \text{ for all } (j, i) \in \mathcal{E}$$

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



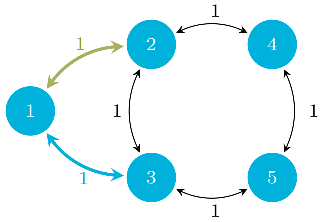
$$\mathbf{A} = \begin{bmatrix} 0 & w_{21} & w_{31} & 0 & 0 \\ w_{12} & 0 & w_{32} & w_{42} & 0 \\ w_{13} & w_{23} & 0 & 0 & w_{53} \\ 0 & w_{24} & 0 & 0 & w_{54} \\ 0 & 0 & w_{35} & w_{45} & 0 \end{bmatrix}.$$

Adjacency Matrices

F. Gama
fgama@berkeley.edu

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

$$[\mathbf{A}]_{ij} = 1, \text{ for all } (j, i) \in \mathcal{E}$$

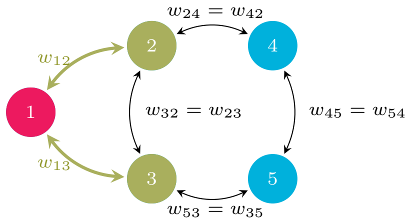


$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Neighborhoods and Degrees

F. Gama
fgama@berkeley.edu

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

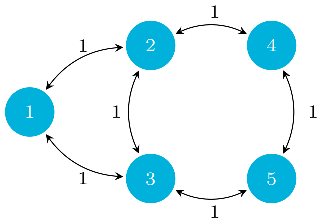


- Node 1 neighborhood $\Rightarrow n(1) = \{2, 3\}$
- Node 1 degree $\Rightarrow n(1) = w_{21} + w_{31}$

Degree Matrix

F. Gama
fgama@berkeley.edu

- The degree matrix is a diagonal matrix $\mathbf{D}$ with degrees as diagonal entries $\Rightarrow [\mathbf{D}]_{ii} = d_i$
- Write in terms of adjacency matrix as $\mathbf{D} = \text{diag}(\mathbf{A}\mathbf{1})$



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

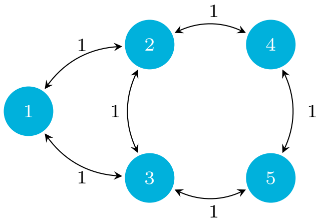
Laplacian Matrix

F. Gama
fgama@berkeley.edu

- The Laplacian matrix of a graph with adjacency matrix $\mathbf{A}$ is $\Rightarrow \mathbf{L} = \mathbf{D} - \mathbf{A} = \mathsf{diag}(\mathbf{A}\mathbf{1}) - \mathbf{A}$
- Can also be written explicitly in terms of graph weights $[\mathbf{A}]_{ij} = w_{ji}$

$\Rightarrow$ Off diagonal entries $\Rightarrow [\mathbf{L}]_{ij} = -[\mathbf{A}]_{ij} = -w_{ji}$

$\Rightarrow$ Diagonal entries $\Rightarrow [\mathbf{L}]_{ii} = d_i = \sum_{j \in n(i)} w_{ji}$



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

Normalized Adjacency and Laplacian Matrices

F. Gama
fgama@berkeley.edu

- Normalized adjacency and Laplacian matrices express weights relative to the nodes' degrees
- Normalized adjacency matrix $\Rightarrow \bar{\mathbf{A}} := \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. It is symmetric if the graph is undirected
- Normalized Laplacian matrix $\Rightarrow \bar{\mathbf{L}} := \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. It is symmetric if the graph is undirected
- Given definitions of Normalized adjacency and Laplacian $\Rightarrow \bar{\mathbf{L}} := \mathbf{I} - \bar{\mathbf{A}}$

Graph Shift Operator

F. Gama
fgama@berkeley.edu

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

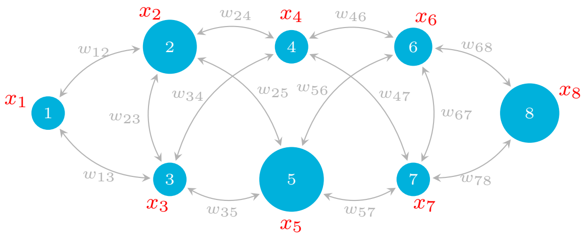
- If the graph is undirected, the shift operator $\mathbf{S}$ is symmetric $\Rightarrow \mathbf{S} = \mathbf{S}^{\mathsf{T}}$

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

Graph Signal

F. Gama
fgama@berkeley.edu

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



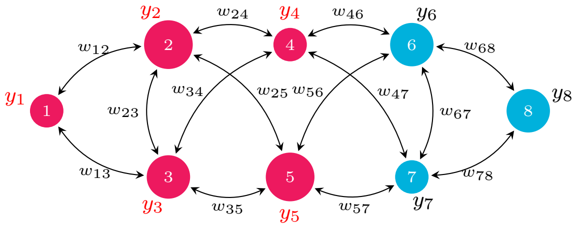
- The graph is an expectation of proximity or similarity between components of the signal $\mathbf{x}$

Graph Signal Diffusion

F. Gama
fgama@berkeley.edu

- Multiplication by the graph shift operator implements diffusion of the signal over the graph
- Diffused signal $\mathbf{y} = \mathbf{S}\mathbf{x}$ has components $\Rightarrow y_i = \sum_{j \in n(i)} w_{ji} x_j$

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



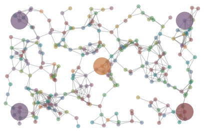
The Diffusion Sequence

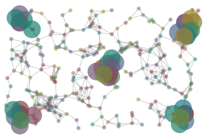
F. Gama
fgama@berkeley.edu

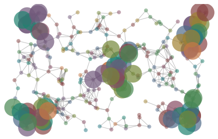
- The shift operator can be composed to produce the diffusion sequence defined recursively as

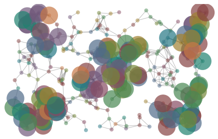
$$\mathbf{x}^{(k+1)} = \mathbf{S}\mathbf{x}^{(k)}, \qquad \text{with} \quad \mathbf{x}^{(0)} = \mathbf{x}$$

- Alternatively, unroll the recursion to write $\Rightarrow \mathbf{x}^{(k)} = \mathbf{S}^k\mathbf{x}$


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


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


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


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

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

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

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

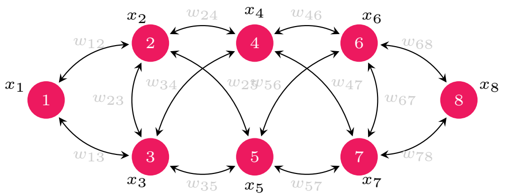
- We say that $\mathbf{y} = \mathbf{h} *_{\mathbf{S}} \mathbf{x}$ is the graph convolution of the filter $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$ with the signal $\mathbf{x}$

F. Gama
fgama@berkeley.edu

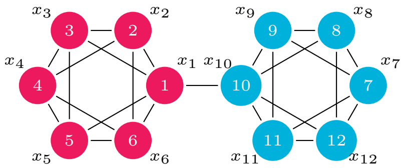
From Local to Global Information

- Graph convolutions aggregate information growing from local to global neighborhoods
  - $\Rightarrow$ Same as regular convolutions in time and space

Graph Filter on a Graph



Same Graph Filter on Another Graph

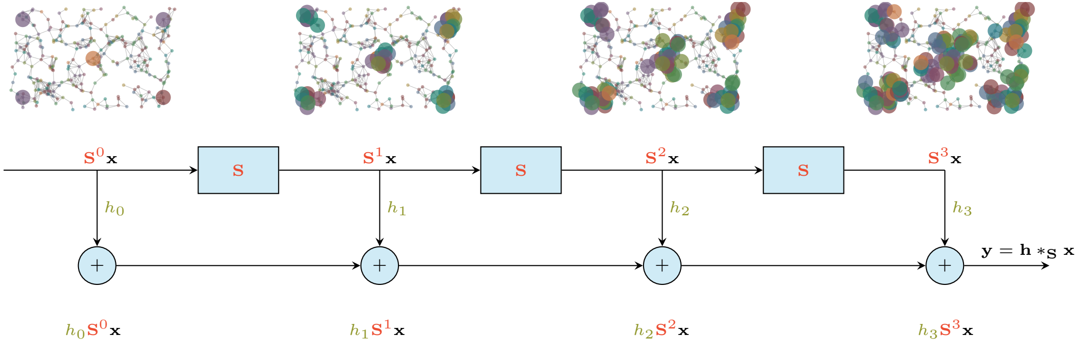


- Filter with coefficients $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$ $\Rightarrow \mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$
- The output of a graph depends on the filter coefficients $\mathbf{h}$ and the graph shift operator $\mathbf{S}$

F. Gama
fgama@berkeley.edu

Graph Convolutional Filters as Diffusion Operators

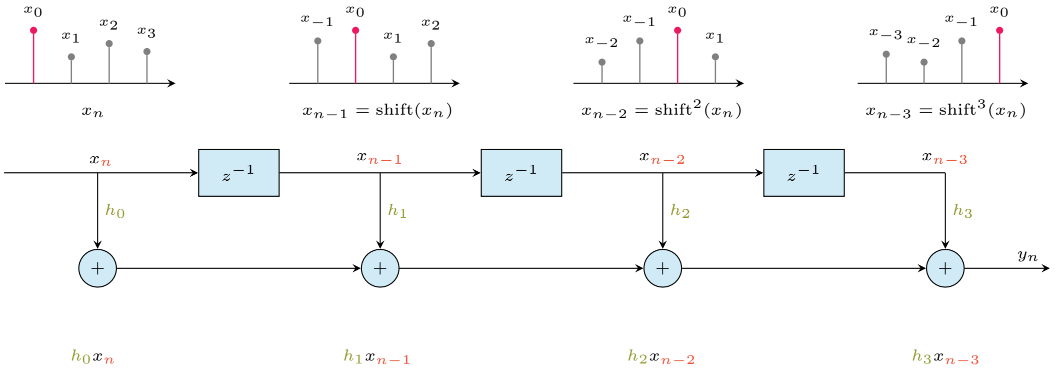
- A graph convolution is a weighted linear combination of the elements of the diffusion sequence
- We can represent the graph convolution with a structure reminiscent of a shift register


$\mathbf{S}^0\mathbf{x}$
$\mathbf{S}$
$\mathbf{S}^1\mathbf{x}$
$\mathbf{S}$
$\mathbf{S}^2\mathbf{x}$
$\mathbf{S}$
$\mathbf{S}^3\mathbf{x}$
h_0
h_1
h_2
h_3
$+$
$\mathbf{y} = \mathbf{h} *_{\mathbf{S}} \mathbf{x}$
$h_0\mathbf{S}^0\mathbf{x}$
$h_1\mathbf{S}^1\mathbf{x}$
$h_2\mathbf{S}^2\mathbf{x}$
$h_3\mathbf{S}^3\mathbf{x}$

Convolutions in Time

F. Gama
fgama@berkeley.edu

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

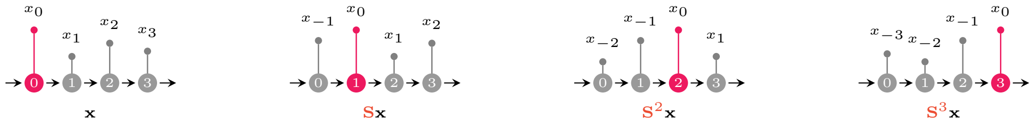

x_0 x_1 x_2 x_3
x_n
x_{-1} x_0 x_1 x_2
$x_{n-1} = \text{shift}(x_n)$
x_{-2} x_{-1} x_0 x_1
$x_{n-2} = \text{shift}^2(x_n)$
x_{-3} x_{-2} x_{-1} x_0
$x_{n-3} = \text{shift}^3(x_n)$
x_n x_{n-1} x_{n-2} x_{n-3}
z^{-1}
h_0 h_1 h_2 h_3
$+$
y_n
$h_0 x_n$ $h_1 x_{n-1}$ $h_2 x_{n-2}$ $h_3 x_{n-3}$

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

Time Signals Represented as Graph Signals

F. Gama
fgama@berkeley.edu

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



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

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

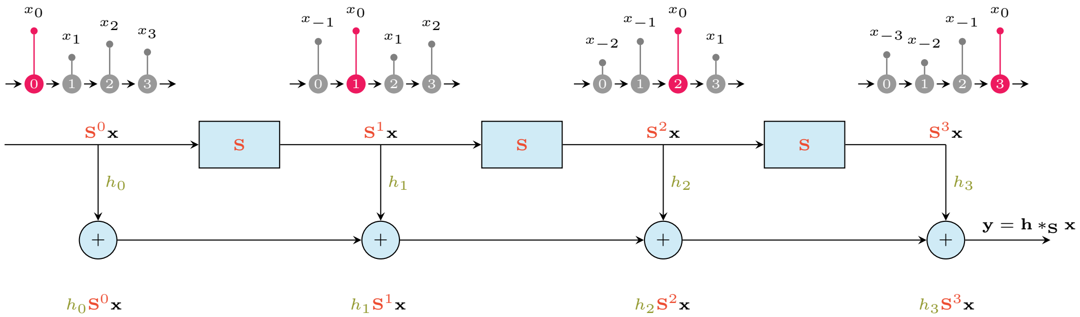
- Components of the shift sequence are powers of the adjacency matrix applied to the original signal

  $\Rightarrow$ We can rewrite convolutional filters as polynomials on $\mathbf{S}$

The Convolution as a Polynomial

F. Gama
fgama@berkeley.edu

- 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

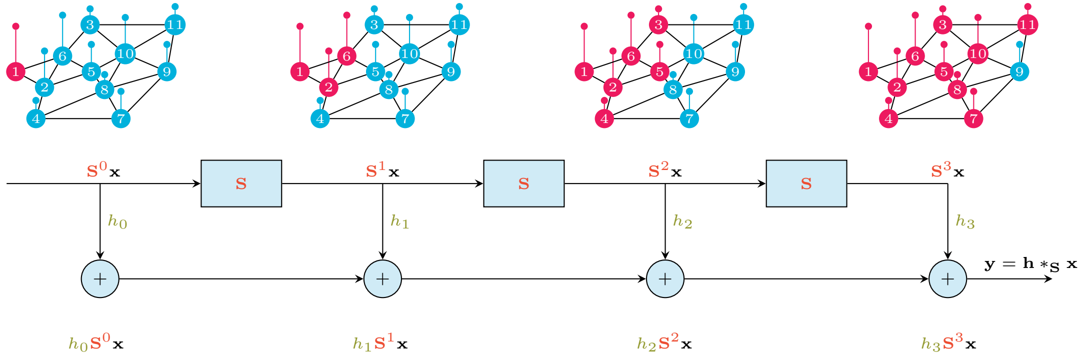

x_0 x_1 x_2 x_3
x_{-1} x_0 x_1 x_2
x_{-2} x_{-1} x_0 x_1
x_{-3} x_{-2} x_{-1} x_0
0 1 2 3
$\mathbf{S}^0\mathbf{x}$
$\mathbf{S}^1\mathbf{x}$
$\mathbf{S}^2\mathbf{x}$
$\mathbf{S}^3\mathbf{x}$
S
h_0 h_1 h_2 h_3
$\mathbf{y} = \mathbf{h} *_{\mathbf{S}} \mathbf{x}$
$h_0\mathbf{S}^0\mathbf{x}$
$h_1\mathbf{S}^1\mathbf{x}$
$h_2\mathbf{S}^2\mathbf{x}$
$h_3\mathbf{S}^3\mathbf{x}$

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

The Time Convolution Generalized to Arbitrary Graphs

F. Gama
fgama@berkeley.edu

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


1
2
3
4
5
6
7
8
9
10
11
$\mathbf{S}^0\mathbf{x}$
$\mathbf{S}^1\mathbf{x}$
$\mathbf{S}^2\mathbf{x}$
$\mathbf{S}^3\mathbf{x}$
$\mathbf{S}$
h_0
h_1
h_2
h_3
$+$
$\mathbf{y} = \mathbf{h} *_{\mathbf{S}} \mathbf{x}$
$h_0\mathbf{S}^0\mathbf{x}$
$h_1\mathbf{S}^1\mathbf{x}$
$h_2\mathbf{S}^2\mathbf{x}$
$h_3\mathbf{S}^3\mathbf{x}$

The Graph Fourier Transform

F. Gama
fgama@berkeley.edu

- Graph filters admit a pointwise representation in a graph frequency domain
  $\Rightarrow$ A property they share with time convolutional filters
- Assume the graph shift operator is normal. E.g., because the graph is undirected $\Rightarrow \mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathsf{H}}$
- The Graph Fourier transform (GFT) of graph signal $\mathbf{x}$ is the signal $\tilde{\mathbf{x}} = \mathbf{V}^{\mathsf{H}}\mathbf{x}$
- The inverse Graph Fourier transform (iGFT) of GFT $\tilde{\mathbf{x}}$ is the graph signal $\mathbf{x} = \mathbf{V}\tilde{\mathbf{x}}$
- We say that the GFT $\tilde{\mathbf{x}}$ is a representation of $\mathbf{x}$ in the graph frequency domain

Graph Convolutions in the Graph Frequency Domain

F. Gama
fgama@berkeley.edu

- Recall definition of graph convolutions as polynomials on the shift operator $\Rightarrow \mathbf{y} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$
- Spectral decomposition of shift operator implies $\mathbf{S}^k = \mathbf{V}\boldsymbol{\Lambda}^k\mathbf{V}^{\mathsf{H}}$. Therefore $\Rightarrow \mathbf{y} = \sum_{k=0}^{\infty} h_k \mathbf{V}\boldsymbol{\Lambda}^k\mathbf{V}^{\mathsf{H}} \mathbf{x}$
- Multiply on the left by $\mathbf{V}^{\mathsf{H}}$. Output GFT $\mathbf{V}^{\mathsf{H}}\mathbf{y} = \tilde{\mathbf{y}}$. Input GFT $\mathbf{V}^{\mathsf{H}}\mathbf{x} = \tilde{\mathbf{x}}$. Cancel out $\mathbf{V}^{\mathsf{H}}\mathbf{V}$

$$\mathbf{V}^{\mathsf{H}}\mathbf{y} = \mathbf{V}^{\mathsf{H}} \sum_{k=0}^{\infty} h_k \mathbf{V}\boldsymbol{\Lambda}^k\mathbf{V}^{\mathsf{H}}\mathbf{x} \qquad \Rightarrow \qquad \tilde{\mathbf{y}} = \left(\sum_{k=0}^{\infty} h_k \boldsymbol{\Lambda}^k\right)\tilde{\mathbf{x}}$$

- The GFTs of the input and output signals are related by a diagonal matrix. $\boldsymbol{\Lambda}$ is diagonal

Graph Frequency Response

F. Gama
fgama@berkeley.edu

- Graph convolutions represented in the graph frequency (spectral) domain $\Rightarrow \tilde{\mathbf{y}} = \left(\sum_{k=0}^{\infty} h_k \mathbf{\Lambda}^k \right) \tilde{\mathbf{x}}$
- Graph convolutions are pointwise in the GFT domain $\Rightarrow \tilde{y}_i = \left(\sum_{k=0}^{\infty} h_k \lambda_i^k \right) \tilde{x}_i$

Definition

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

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

- Defined so that we can write $\Rightarrow \tilde{y}_i = \tilde{\mathsf{h}}(\lambda_i) \tilde{x}_i$

Graph Frequency Response

F. Gama
fgama@berkeley.edu

Definition

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

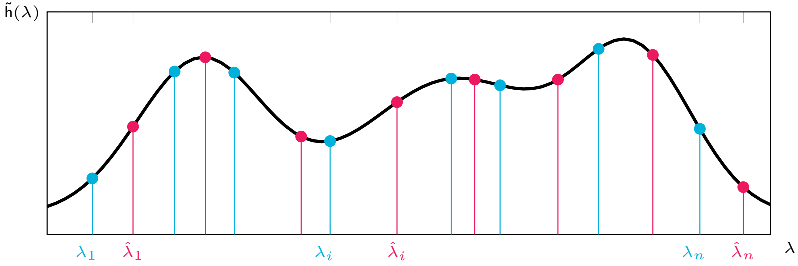
$$\tilde{\mathsf{h}}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$$

- The frequency response is the same polynomial that defines the graph filter but on scalar variable λ
- Frequency response is independent of the graph. It is completely determined by the filter coefficients
- The role of the graph is to determine the eigenvalues on which the response is instantiated

Effect of Running the Same Filter on Different Graphs

F. Gama
fgama@berkeley.edu

- Given coefficients $\mathbf{h} = \{h_k\}_{k=1}^{\infty}$ the graph filter frequency response is determined $\tilde{\mathsf{h}}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$
- For a specific graph, the response is instantiated on its specific eigenvalues λ_i
- For a different graph, the response is instantiated different eigenvalues $\hat{\lambda}_i$

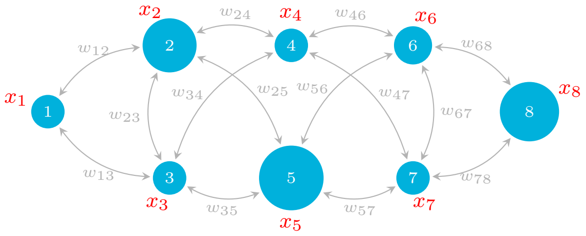


- Key to graph perturbations analyses $\Rightarrow$ Instrumental in explaining good performance of GNNs

Memorabilia: Graphs and Graph Signals

F. Gama
fgama@berkeley.edu

- For a given graph $\mathcal{G}$, a graph shift operator $\mathbf{S} \in \mathbb{R}^{N \times N}$ is a matrix representation of a graph
- A graph signal $\mathbf{x} \in \mathbb{R}^N$ is a vector with component x_i associated to node i
- If we write $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathsf{H}}$, the graph Fourier transform (GFT) of signal $\mathbf{x}$ is $\tilde{\mathbf{x}} = \mathbf{V}^{\mathsf{H}}\mathbf{x}$

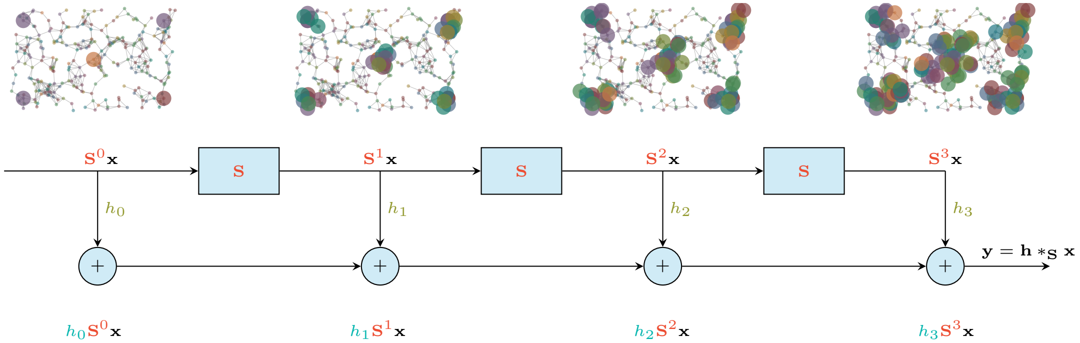

x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
1 2 3 4 5 6 7 8
w_{12} w_{13} w_{23} w_{24} w_{25} w_{34} w_{35} w_{46} w_{47} w_{56} w_{57} w_{67} w_{68} w_{78}

Memorabilia: Graph Filters

F. Gama
fgama@berkeley.edu

- Graph filter with coefficients $\mathbf{h} = \{h_k\}_{k=1}^{\infty}$ is a polynomial on shift operator $\Rightarrow \mathbf{H}(\mathbf{S}) = \sum_{k=0}^{\infty} h_k \mathbf{S}^k$

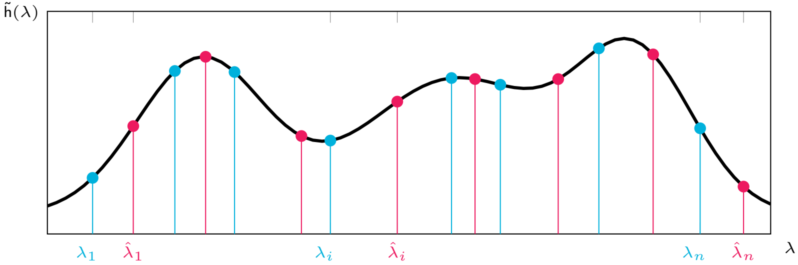
- A graph convolution is the output of a graph filter $\Rightarrow \mathbf{y} = \mathbf{h} *_{\mathbf{S}} \mathbf{x} = \mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$


$\mathbf{S}^0\mathbf{x}$
$\mathbf{S}$
$\mathbf{S}^1\mathbf{x}$
$\mathbf{S}$
$\mathbf{S}^2\mathbf{x}$
$\mathbf{S}$
$\mathbf{S}^3\mathbf{x}$
h_0
h_1
h_2
h_3
$+$
$\mathbf{y} = \mathbf{h} *_{\mathbf{S}} \mathbf{x}$
$h_0\mathbf{S}^0\mathbf{x}$
$h_1\mathbf{S}^1\mathbf{x}$
$h_2\mathbf{S}^2\mathbf{x}$
$h_3\mathbf{S}^3\mathbf{x}$

Memorabilia: Graph Frequency Response of a Graph Filter

F. Gama
fgama@berkeley.edu

- Graph frequency response of a graph filter is polynomial on scalar variable $\Rightarrow \tilde{\mathsf{h}}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$
- The GFT domain input-output relationship of a graph filter is pointwise $\Rightarrow \tilde{y}_i = \tilde{\mathsf{h}}(\lambda_i)\tilde{x}_i$
- Response depends on filter only. It's the same for all graphs. Graph instantiates specific eigenvalues



Graph Neural Networks

F. Gama
fgama@berkeley.edu

Empirical Risk Minimization

F. Gama
fgama@berkeley.edu

- In this short course, machine learning $\equiv$ empirical risk minimization (ERM).
- In ERM we are given:
  - $\Rightarrow$ A training set $\mathcal{T}$ containing observation pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}$. Assume equal dimension $\mathbf{x}, \mathbf{y}, \in \mathbb{R}^N$.
  - $\Rightarrow$ A loss function $\mathsf{J}(\mathbf{y}, \hat{\mathbf{y}})$ to evaluate the similarity between $\mathbf{y}$ and an estimate $\hat{\mathbf{y}}$
  - $\Rightarrow$ A function class $\mathbf{\Phi}$
- Learning means finding function $\Phi^\star \in \mathbf{\Phi}$ that minimizes loss $\mathsf{J}\Big(\Phi(\mathbf{x}), \mathbf{y}\Big)$ averaged over training set

$$\Phi^\star = \underset{\Phi \in \mathbf{\Phi}}{\arg\min} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \mathsf{J}\Big(\Phi(\mathbf{x}), \mathbf{y}\Big)$$

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

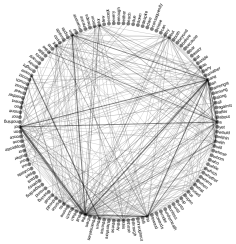
Empirical Risk Minimization

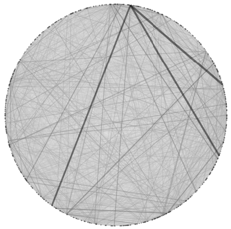
F. Gama
fgama@berkeley.edu

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

$$\Phi^{\star} = \underset{\Phi \in \mathbf{\Phi}}{\arg\min} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \mathsf{J}\Big(\mathbf{\Phi}(\mathbf{x}), \mathbf{y}\Big)$$

- Designing a Machine Learning algorithm $\equiv$ finding the right function class $\mathbf{\Phi}$
- Since we are interested in graph signals, graph convolutional filters look like a good starting point

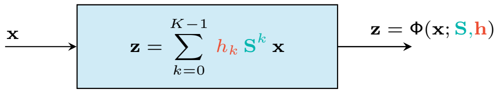




Learning with a Convolutional Graph Filter

F. Gama
fgama@berkeley.edu

- Assume the input signals $\mathbf{x}$ are graph signals supported on a common graph with shift operator $\mathbf{S}$
- Choose function class as graph filters of order K supported on $\mathbf{S}$ $\Rightarrow \boldsymbol{\Phi}(\mathbf{x}) = \boldsymbol{\Phi}(\mathbf{x}; \mathbf{S}, \mathbf{h}) = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$


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

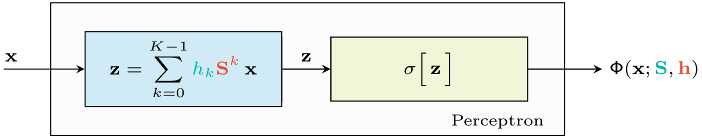
- Learn ERM solution restricted to graph filter class $\Rightarrow \mathbf{h}^\star = \arg\min_{\mathbf{h}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \mathsf{J}\Big(\boldsymbol{\Phi}(\mathbf{x}; \mathbf{S}, \mathbf{h}), \mathbf{y}\Big)$

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

Learning with a Graph Perceptron

F. Gama
fgama@berkeley.edu

- Graph filters have limited expressive power because they can only learn linear maps
- An alternative choice for $\mathbf{\Phi}$ is the set of graph perceptrons $\Rightarrow \Phi(\mathbf{x}) = \Phi(\mathbf{x};\mathbf{S},\mathbf{h}) = \sigma\left[\sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}\right]$


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

- Optimal regressor restricted to perceptron class $\Rightarrow \mathbf{h}^\star = \arg\min_{\mathbf{h}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\Phi(\mathbf{x};\mathbf{S},\mathbf{h}),\mathbf{y}\Big)$

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

Graph Neural Networks (GNNs)

F. Gama
fgama@berkeley.edu

- We increase expressivity further with a GNN

- Layer a few graph perceptrons (3 in the figure)

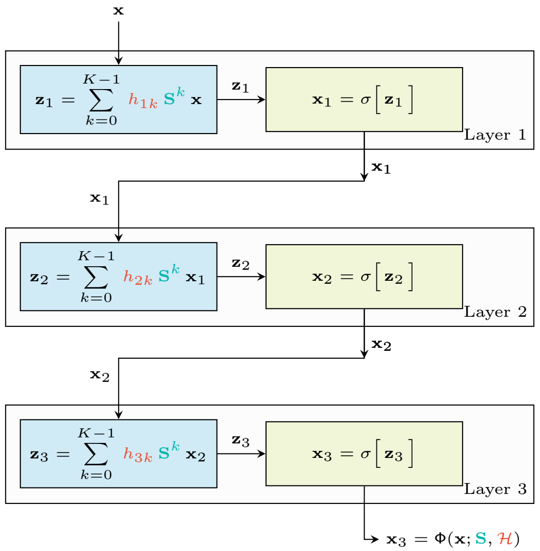
  $\Rightarrow$ Feed the input signal $\mathbf{x}$ to Layer 1

  $\Rightarrow$ Connect output of Layer 1 to input of Layer 2

  $\Rightarrow$ And output of Layer 2 to input of Layer 3

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

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


$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k} \mathbf{S}^k \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma[\mathbf{z}_1]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k} \mathbf{S}^k \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma[\mathbf{z}_2]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k} \mathbf{S}^k \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma[\mathbf{z}_3]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

Learning with a Graph Neural Network

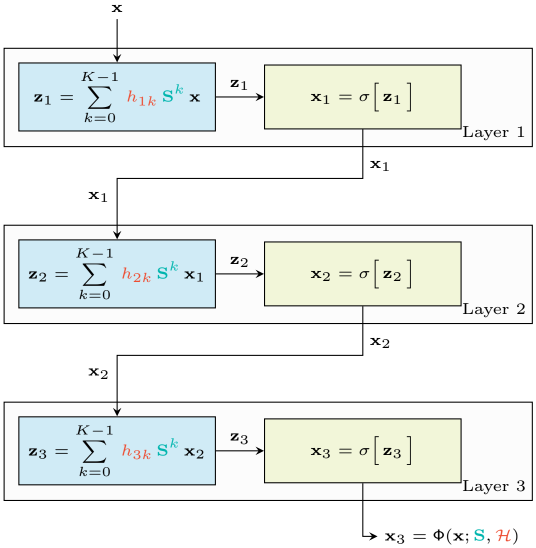
F. Gama
fgama@berkeley.edu

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

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

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

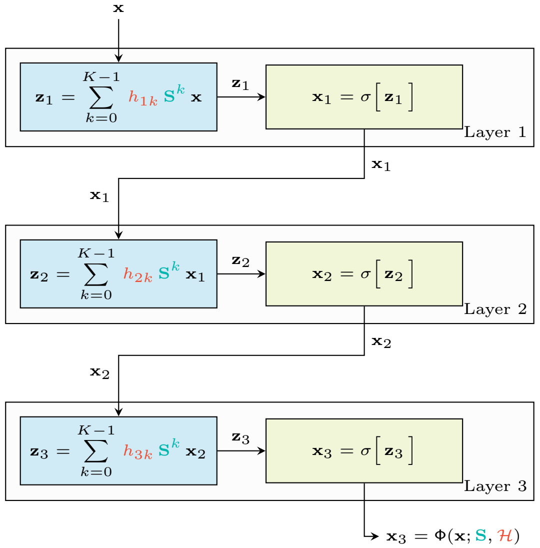
$\Rightarrow$ Prior information given to the GNN


$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k}\,\mathbf{S}^k\,\mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\big[\mathbf{z}_1\big]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k}\,\mathbf{S}^k\,\mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\big[\mathbf{z}_2\big]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k}\,\mathbf{S}^k\,\mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\big[\mathbf{z}_3\big]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x};\mathbf{S},\mathcal{H})$

Graph Neural Networks and Graph Filters

F. Gama
fgama@berkeley.edu

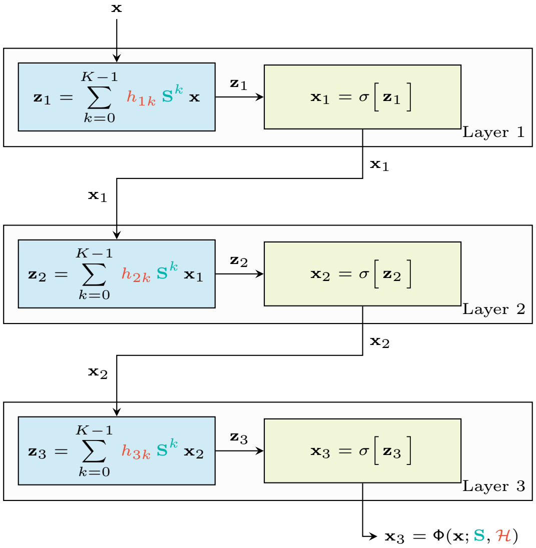
- GNNs are minor variations of graph filters
- Only difference is adding pointwise nonlinearities
  - $\Rightarrow$ Signal entries are processed individually
  - $\Rightarrow$ There is no mixing of components
- GNNs do work (much) better than graph filters
  - $\Rightarrow$ Which is surprising and deserves explanation
  - $\Rightarrow$ Which we will attempt with stability analyses


$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k}\,\mathbf{S}^k\,\mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\big[\,\mathbf{z}_1\,\big]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k}\,\mathbf{S}^k\,\mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\big[\,\mathbf{z}_2\,\big]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k}\,\mathbf{S}^k\,\mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\big[\,\mathbf{z}_3\,\big]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

Transference of GNNs Across Graphs

F. Gama
fgama@berkeley.edu

- GNN Output depends on the graph $\mathbf{S}$.
- We can think of $\mathbf{S}$ as a parameter
  - $\Rightarrow$ Encoding prior information. As we've done so far
- Or, we can reinterpret $\mathbf{S}$ as an input
  - $\Rightarrow$ Enabling transference across graphs
  - $\Rightarrow$ A trained GNN is just a filter tensor $\mathcal{H}^\star$

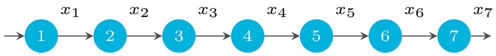

$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k}\, \mathbf{S}^k\, \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\big[\mathbf{z}_1\big]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k}\, \mathbf{S}^k\, \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\big[\mathbf{z}_2\big]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k}\, \mathbf{S}^k\, \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\big[\mathbf{z}_3\big]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

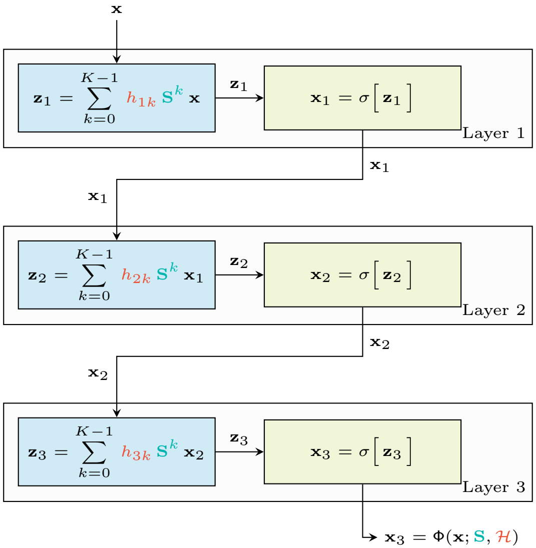
CNNs and GNNs

F. Gama
fgama@berkeley.edu

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

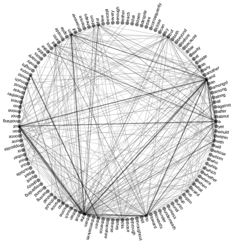
$$\mathbf{S} = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & 0 & 0 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & \cdots \\ & \vdots & \vdots & \vdots & \end{bmatrix}$$

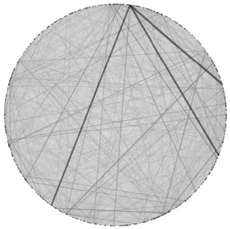



$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k}\,\mathbf{S}^k\,\mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\left[\mathbf{z}_1\right]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k}\,\mathbf{S}^k\,\mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\left[\mathbf{z}_2\right]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k}\,\mathbf{S}^k\,\mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\left[\mathbf{z}_3\right]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

The Road Not Taken: Fully Connected Neural Networks

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

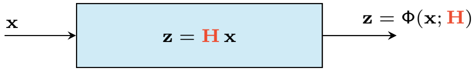




Learning with a Linear Classifier

F. Gama
fgama@berkeley.edu

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

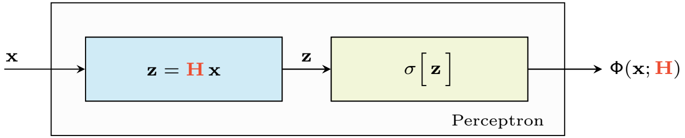

$\mathbf{x}$
$\mathbf{z} = \mathbf{H}\,\mathbf{x}$
$\mathbf{z} = \Phi(\mathbf{x}; \mathbf{H})$

- Optimal regressor is ERM solution restricted to linear class $\Rightarrow \mathbf{H}^\star = \arg\min_{\mathbf{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} J\Big(\Phi(\mathbf{x}; \mathbf{H}), \mathbf{y}\Big)$

Learning with a Linear Perceptron

F. Gama
fgama@berkeley.edu

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

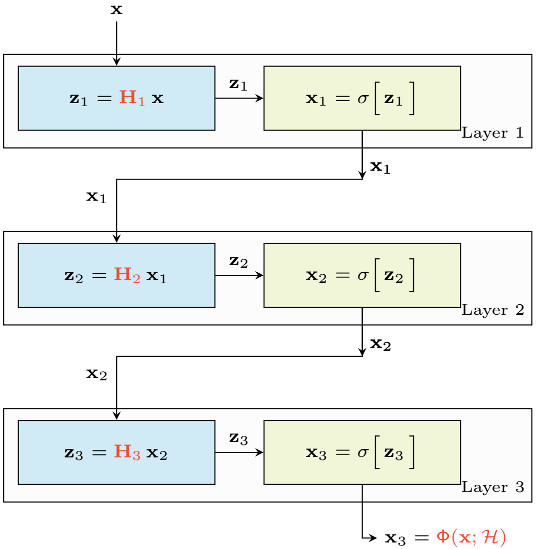

$\mathbf{x}$
$\mathbf{z} = \mathbf{H}\,\mathbf{x}$
$\mathbf{z}$
$\sigma\Big[\mathbf{z}\Big]$
$\Phi(\mathbf{x};\mathbf{H})$
Perceptron

- Optimal regressor restricted to perceptron class $\Rightarrow \mathbf{H}^{\star} = \underset{\mathbf{H}}{\arg\min} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\Phi(\mathbf{x};\mathbf{H}),\mathbf{y}\Big)$

Fully Connected Neural Network (NN)

F. Gama
fgama@berkeley.edu

- Further increase expressivity with a NN
- Layer a few graph perceptrons (3 in the figure)
  - ⇒ Feed the input signal $\mathbf{x}$ to Layer 1
  - ⇒ Connect output of Layer 1 to input of Layer 2
  - ⇒ And output of Layer 2 to input of Layer 3
- Last layer output is the NN output $\Rightarrow \Phi(\mathbf{x}; \mathcal{H})$
  - ⇒ Parametrized on tensor $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$


$\mathbf{x}$
$\mathbf{z}_1 = \mathbf{H}_1 \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\left[\mathbf{z}_1\right]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \mathbf{H}_2 \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\left[\mathbf{z}_2\right]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \mathbf{H}_3 \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\left[\mathbf{z}_3\right]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathcal{H})$

Which is Better, a GNN or an Arbitrary NN?

F. Gama
fgama@berkeley.edu

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

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

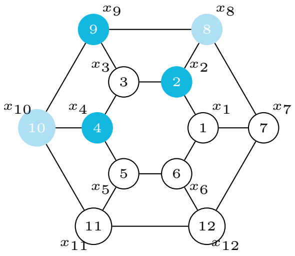
- The fully connected NN does better. But this holds for the training set
- In practice, the GNN does better because it generalizes better to unseen signals

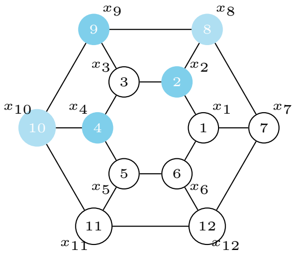
$\Rightarrow$ Because it exploits internal symmetries of graph signals codified in the graph shift operator

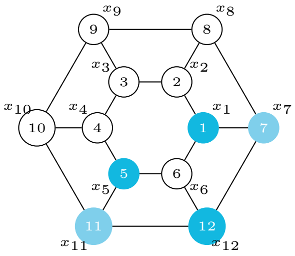
F. Gama
fgama@berkeley.edu

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



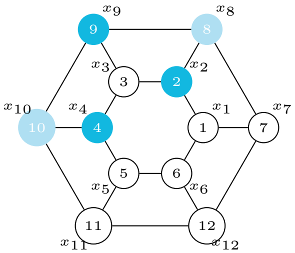


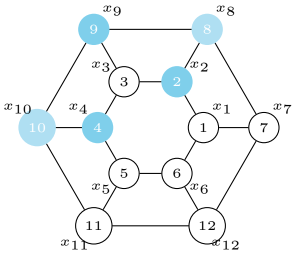


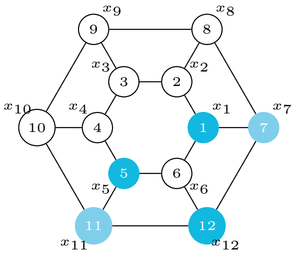
Generalization with a Graph Neural Network

F. Gama
fgama@berkeley.edu

- 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
- The GNN still learns how to fill the middle signal. But it also learns how to fill the right signal



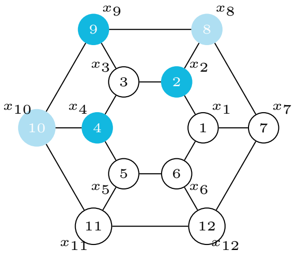


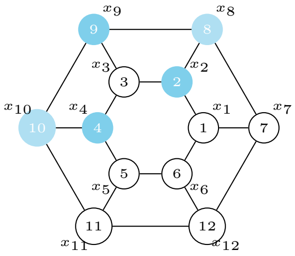


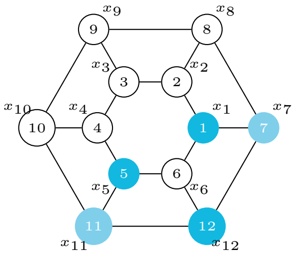
Permutation Equivariance of Graph Neural Network

F. Gama
fgama@berkeley.edu

- This will be formalized later as the permutation equivariance of graph neural networks
- It is analogous to the shift equivariance of convolutional neural networks. Which is a particular case





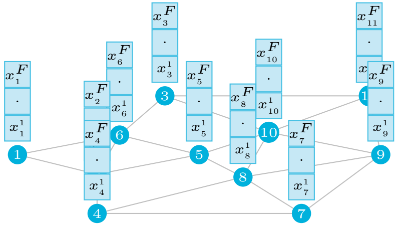


Multiple Features and Pooling

F. Gama
fgama@berkeley.edu

Multiple Features

- Graph signal $\Rightarrow$ Single scalar associated to each node $\Rightarrow \mathbf{x}: \text{nodes} \to \mathbb{R}$
- Extend descriptive power $\Rightarrow$ Assign a vector to each node
  $\Rightarrow \mathbf{X}: \text{nodes} \to \mathbb{R}^2 \Rightarrow \mathbf{X}: \text{nodes} \to \mathbb{R}^3 \Rightarrow \mathbf{X}: \text{nodes} \to \mathbb{R}^F$
- Multiple feature graph signal $\Rightarrow$ Matrix $\mathbf{X}$ of size N (nodes) $\times$ F (features)
  $\Rightarrow$ Row i collects all features at node i $\Rightarrow$ Local information at node i
  $\Rightarrow$ Column f represents feature f at all nodes $\Rightarrow \mathbf{x}^f$ is a graph signal



$$\mathbf{X} = \begin{bmatrix} x_1^1 & x_1^2 & x_1^3 & \cdots & x_1^F \\ x_2^1 & x_2^2 & x_2^3 & \cdots & x_2^F \\ x_3^1 & x_3^2 & x_3^3 & \cdots & x_3^F \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{10}^1 & x_{10}^2 & x_{10}^3 & \cdots & x_{10}^F \\ x_{11}^1 & x_{11}^2 & x_{11}^3 & \cdots & x_{11}^F \end{bmatrix}$$

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2019.

Extending Convolution

F. Gama
fgama@berkeley.edu

- Convolution $\Rightarrow$ Linear operation, local information, distributed implementation

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

- Multiplication by $\mathbf{S}$ on the left $\Rightarrow$ Shifts each graph signal feature $\mathbf{S}\mathbf{x}^f$
- Multiplication by $\mathbf{H}$ on the right $\Rightarrow$ Linear combination of features at each node (No exchanges)

- The convolution is now equivalent to the application of a bank of graph filters
- For each feature $\mathbf{x}^f$ we apply the (f, g) filter to obtain $\mathbf{x}^{fg}$ $\Rightarrow$ Linear, local, distributed

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

$\Rightarrow$ There are G filters applied to each input $\mathbf{x}^f$ $\Rightarrow$ There a total of $F \times G$ filters (size of $\mathbf{H}_k$)

- Aggregate the output of each gth filter across all F features $\Rightarrow \mathbf{y}^g = \sum_{f=1}^{F} \mathbf{x}^{fg} \Rightarrow \mathbf{Y} = [\mathbf{y}^1, \ldots, \mathbf{y}^G]$

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2019

Graph Neural Networks

- Single feature graph neural networks $\Rightarrow \boldsymbol{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}) = \mathbf{x}_L$

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

$\mathbf{x}_\ell$ graph signal at layer ℓ $\Rightarrow$ Size $N \times 1$

$h_{\ell k}$ kth filter tap at layer ℓ $\Rightarrow$ Size 1×1 $\Rightarrow$ Learn $\sum_\ell K_\ell$ filter taps $\mathcal{H} = \{h_{\ell k}\}$

$\mathbf{S}$ shift operator (given by the problem) $\Rightarrow$ Size $N \times N$

- Hyperparameters (design choices) $\Rightarrow$ Affect the number of learnable parameters

$\Rightarrow$ L: number of layers

$\Rightarrow$ K_ℓ: number of filter taps on each layer

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2019

Graph Neural Networks

- Multi feature graph neural networks $\Rightarrow \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H}) = \mathbf{X}_L$

$$\mathbf{X}_\ell = \sigma\Big[\sum_{k=0}^{K_\ell - 1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\Big]$$

$\mathbf{X}_\ell$ graph signal at layer ℓ $\Rightarrow$ Size $N \times F_\ell$

$\mathbf{H}_{\ell k}$ kth filter tap at layer ℓ $\Rightarrow$ Size $F_{\ell-1} \times F_\ell$ $\Rightarrow$ Learn $\sum_\ell K_\ell F_\ell F_{\ell-1}$ filter taps $\mathcal{H} = \{\mathbf{H}_{\ell k}\}$

$\mathbf{S}$ shift operator (given by the problem) $\Rightarrow$ Size $N \times N$

- Hyperparameters (design choices) $\Rightarrow$ Affect the number of learnable parameters

$\Rightarrow$ L: number of layers

$\Rightarrow$ K_ℓ: number of filter taps on each layer

$\Rightarrow$ F_ℓ: number of features on each layer

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2019

F. Gama
fgama@berkeley.edu

Pooling

- Increasing number of features in each layer $\Rightarrow$ Increasing dimension of the signals

  $\Rightarrow$ Increasing computational cost of the convolution operation

- Pooling $\mathcal{P}$ $\Rightarrow$ Construct regional summaries of information at each layer

$$\mathbf{X}_\ell = \mathcal{P}_\ell \left\{ \sigma \left[\sum_{k=0}^{K_\ell - 1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k} \right] \right\}$$

  $\Rightarrow$ Keep only some summaries $\Rightarrow$ $\mathbf{X}_\ell$ of size $N_\ell \times F_\ell$ with $N_\ell \leq N_{\ell-1} \leq N$

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2019

Ruiz, Gama, Marques, Ribeiro, "Invariance-Preserving Localized Activation Functions for Graph Neural Networks", IEEE TSP 2020

Summarizing and Sampling

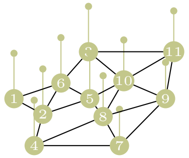
F. Gama
fgama@berkeley.edu

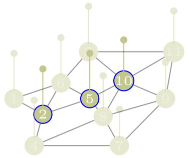
- Pooling $\Rightarrow$ Local summarizing function followed by downsampling

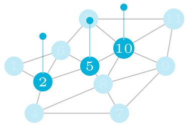
$$\mathbf{z}_\ell = \mathcal{P}_\ell\Big\{\mathbf{z}_{\ell-1}\Big\} = \mathbf{C}_\ell\,\rho\Big[\mathbf{z}_{\ell-1};\mathbf{S}\Big]$$

$\Rightarrow$ $\rho[\cdot;\mathbf{S}]$ summarizing function $\Rightarrow$ Example $[\rho[\mathbf{z};\mathbf{S}]]_i = \max\{[\mathbf{z}]_i, [\mathbf{S}\mathbf{z}]_i, \ldots, [\mathbf{S}^{K-1}\mathbf{z}]_i\}$

$\Rightarrow$ $\mathbf{C}_\ell$ binary $N_\ell \times N_{\ell-1}$ selection matrix $\Rightarrow$ $[\mathbf{C}_\ell]_{ij} \in \{0,1\}$, $\mathbf{C}\mathbf{1} = \mathbf{1}$, $\mathbf{C}^\mathsf{T}\mathbf{1} \preceq \mathbf{1}$







Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2019

Ruiz, Gama, Marques, Ribeiro, "Invariance-Preserving Localized Activation Functions for Graph Neural Networks", IEEE TSP 2020

Sampling and Convolution

- Graph convolution $\Rightarrow$ Shift operator respects the size of the graph $N \times N$

$$\mathbf{C}_{\ell-1}\mathbf{Y}_{\ell} = \sum_{k=0}^{K-1} \mathbf{C}_{\ell-1}\mathbf{S}^{k}\mathbf{C}_{\ell-1}^{\mathsf{T}}\mathbf{X}_{\ell-1} \ \mathbf{H}_{\ell k}$$

- After pooling (sampling) the dimension of the signal $\mathbf{X}_{\ell-1}$ has been reduced $N_{\ell-1} \leq N$

  $\Rightarrow$ Dimension mismatch between $\mathbf{S}$ of size $N \times N$ and $\mathbf{X}_{\ell-1}$ of size $N_{\ell-1} \times F_{\ell-1}$
- Zero padding $\Rightarrow$ Use sampling matrix $\mathbf{C}_{\ell-1}$ to adapt dimensions

  $\Rightarrow$ $\mathbf{C}_{\ell-1}^{\mathsf{T}}\mathbf{X}_{\ell-1}$ has size $N \times F_{\ell-1}$ where unsampled nodes have now a zero value
- Output $\mathbf{Y}_{\ell}$ has dimension N $\Rightarrow$ Needs to be downsampled to have size $N_{\ell-1}$ (same as $\mathbf{X}_{\ell}$)

  $\Rightarrow$ $\mathbf{C}_{\ell-1}\mathbf{Y}_{\ell-1}$ samples the output at the same nodes as $\mathbf{X}_{\ell-1}$ $\Rightarrow$ $\mathbf{C}_{\ell-1}\mathbf{Y}_{\ell-1}$ has size $N_{\ell-1} \times F_{\ell}$

Gama, Marques, Leus, Ribeiro, “Convolutional Neural Network Architectures for Signals Supported on Graphs”, IEEE TSP 2019

Sampling and Convolution: Implementation

F. Gama
fgama@berkeley.edu

- Pooling $\Rightarrow$ Reduce computational complexity of the convolution operation

$$\mathbf{C}_{\ell-1}\mathbf{Y}_{\ell} = \sum_{k=0}^{K-1} \quad \mathbf{S}_{\ell-1}^{(k)} \quad \mathbf{X}_{\ell-1} \ \mathbf{H}_{\ell k}$$

- Centralized computation $\Rightarrow \mathbf{S}_{\ell-1}^{(k)} = \mathbf{C}_{\ell-1}\mathbf{S}^{k}\mathbf{C}_{\ell-1}^{\mathsf{T}}$ reduced kth shift operator $\Rightarrow$ Size $N_{\ell-1} \times N_{\ell-1}$

  $\Rightarrow$ Operations are now on dimensions $N_{\ell-1} \leq N$ $\Rightarrow$ Matrix $\mathbf{S}_{\ell-1}^{(k)}$ is computed before training
- Decentralized computation $\Rightarrow \mathbf{C}_{\ell-1}^{\mathsf{T}}\mathbf{X}_{\ell-1}$ is a graph signal with zeros in unsampled nodes

  $\Rightarrow$ Multiplication of shift operator and graph signal $\mathbf{S}^{k}(\mathbf{C}_{\ell-1}^{\mathsf{T}}\mathbf{X})$ $\Rightarrow$ Local and distributed
- Pooling that reduces computational cost and respects the topology of the graph

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2019

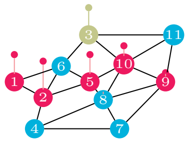
Recommendation Systems

F. Gama
fgama@berkeley.edu

Recommendation Systems

F. Gama
fgama@berkeley.edu

- Items within a collection are being rated by a pool of users
  - $\Rightarrow$ Some pairs of items have been rated by multiple users
  - $\Rightarrow$ Likelihood of two items getting similar ratings
  - $\Rightarrow$ Estimate the Pearson correlation between items
  - $\Rightarrow$ Items are nodes and rating similarities are edges
- A user rates some of the items $\Rightarrow$ Graph signal



We want to know what rating the user would give to a target item

Huang, Marques, Ribeiro, "Rating Prediction via Graph Signal Processing", IEEE TSP 2018

Recommendation Systems: Modeling in Graph Signal Processing

F. Gama
fgama@berkeley.edu

- We want to know what rating x_t the user would give to a target item t $\Rightarrow$ Estimate $[\mathbf{x}]_t = x_t$
  - $\Rightarrow$ Based on the ratings the user has given to some of the items $\Rightarrow$ Graph signal $\mathbf{x}$
  - $\Rightarrow$ Leveraging the rating similarities between items $\Rightarrow$ Graph structure $\mathbf{S}$
- Entry $[\mathbf{x}]_i = 0$ if item i has not been rated. $[\mathbf{S}]_{ij}$ Pearson correlation (k-NN sparsification)
- Train GNN $\Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$ to map between ratings $\mathbf{x}$ of user and estimated rating x_t of item t
  - $\Rightarrow$ Select number of layers L, number of features F_ℓ, number of filter taps K_ℓ
  - $\Rightarrow$ Last layer has $K_L = 1$ and $F_L = 1$ $\Rightarrow$ Extract resulting value $[\mathbf{x}_L]_t = x_t$ at node t

Gama, Isufi, Leus, Ribeiro, "Graphs, Convolutions, and Neural Networks: From Graph Filters to Graph Neural Networks", IEEE SPM 2020

Recommendation Systems: Dataset

- `MovieLens-100k` $\Rightarrow$ 100,000 ratings given by 943 users to 1,582 movies (Ratings go from 1 to 5)
  $\Rightarrow$ Each movie is a node in the graph $\Rightarrow$ 1,582 nodes $\Rightarrow$ `graphType = 'movie'`
- Target movie is *Star Wars* (`labelID = [50]`) $\Rightarrow$ 583 users have rated the movie $\Rightarrow$ Graph signals
  $\Rightarrow$ The actual rating of the target movie x_t is extracted as a label $y = x_t$ $\Rightarrow$ Dataset $\{(\mathbf{x}, y)\}$
- Use `ratioTrain = 0.9` for training set and further split `ratioValid = 0.1` for validation
- Build graph from data $\Rightarrow$ Edges are Pearson correlation $\Rightarrow$ `kNN = 10` nearest neighbors

```python
import Utils.dataTools as dataTools
# Load the data
data = dataTools.MovieLens(graphType, # 'user' or 'movie'
                           labelID, # ID of target node to estimate
                           ratioTrain, # ratio of training samples
                           ratioValid, # ratio of validation samples (out of training samples)
                           dataDir,
                           # Graph building options
                           keepIsolatedNodes, # If True keeps isolated nodes
                           forceUndirected, # If True creates an undirected graph
                           forceConnected, # If True keeps the largest connected component
                           kNN) # Number of nearest neighbors
```

https://github.com/alelab-upenn/graph-neural-networks

F. Gama
fgama@berkeley.edu

Recommendation Systems: Initialize Architecture

- Set the architecture parameters $\Rightarrow L = 2, F_1 = 64, F_2 = 32, K_1 = K_2 = K = 5$, ReLU nonlinearity

```
F1 = 64; F2 = 32; K = 5; nonlinearity = nn.ReLU
```

- Get the shift operator **S** obtained from the dataset $\Rightarrow$ Normalize it

```
S = data.getGraph()
S = S/np.max(np.real(np.linalg.eigvals(S)))
```

- Create the architecture $\Rightarrow$ Use module `architectures.py` provided

```
import Modules.architectures as archit
Phi = archit.LocalGNN(# Graph filtering features (a list element per layer)
                      [1, F1, F2], # 1 is the number of input features at the first layer
                      # Graph filtering taps (a list element per layer)
                      [K, K], # Same number of filter taps in each layer
                      True, # If True include bias vector
                      # Nonlinear function
                      nonlinearity,
                      # Pooling definition (not used)
                      [S.shape[0],S.shape[0]], gml.NoPool, [1,1],
                      # Local readout layer (maps F2 into this given number of output features)
                      [1], # Needs to be a scalar because it represents the estimated rating
                      # Graph shift operator
                      S)
```

https://github.com/alelab-upenn/graph-neural-networks

Recommendation Systems: Training

F. Gama
fgama@berkeley.edu

- `model.py` contains the `Model` class that binds together all elements of a ML model

```
import Modules.model as model
modelGNN = model.Model(Phi, # Architecture
                       nn.SmoothL1Loss(), # Loss function
                       optim.Adam(Phi.parameters(), lr = 0.005), # Optimizer
                       training.TrainerSingleNode, # Trainer
                       evaluation.evaluateSingleNode, # Evaluator
                       'cpu', # Device (either 'cpu' or 'cuda')
                       'LocalGNN', # Name of the architecture
                       saveDir) # Directory where to save the run
```

- The model class has a method that trains the architecture $\Rightarrow$ `nEpochs = 40`, `batchSize = 5`

```
modelGNN.train(data, nEpochs, batchSize)
```

https://github.com/alelab-upenn/graph-neural-networks

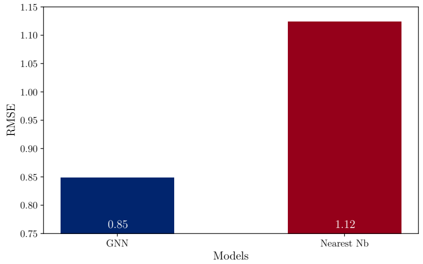
Recommendation Systems: Evaluation

F. Gama
fgama@berkeley.edu

- The `Model` class has a method for evaluation as well

```
evalVars = modelGNN.evaluate(data)
```

- `evalVars` is a dictionary containing the RMSE achieved by the model
  - ⇒ `'costBest'` gives the RMSE (cost) for the parameters that got the lowest validation error
  - ⇒ `'costLast'` gives the RMSE (cost) for the parameters at the last training step



https://github.com/alelab-upenn/graph-neural-networks

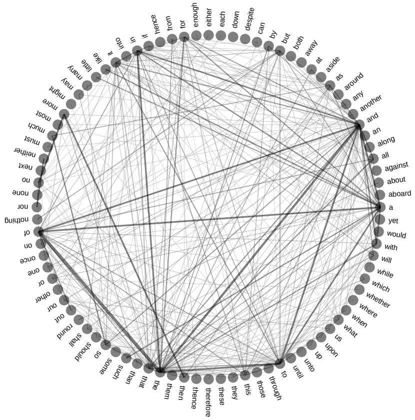
Authorship Attribution

F. Gama
fgama@berkeley.edu

Word Adjacency Networks

F. Gama
fgama@berkeley.edu

- Function words are those that don't carry meaning
- Their use depends on the language's grammar
- Different authors use slightly different grammar
- Capture with a word adjacency network (WAN)
  - ⇒ How often pairs of words appear together

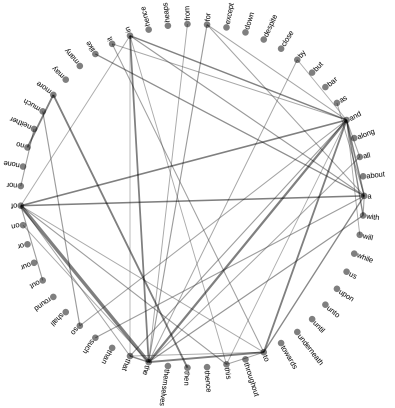


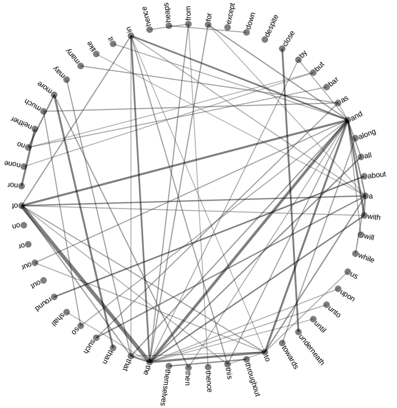
Segarra, Eisen, Ribeiro, "Authorship Attribution through Function Word Adjacency Networks", IEEE TSP 2015

Word Adjacency Networks: Shakespeare or Marlowe?

F. Gama
fgama@berkeley.edu

- Shakespeare's and Marlowe's WANs are sufficiently different to ascertain their collaboration on Henry VI





Segarra, Eisen, Egan, Ribeiro, "Attributing the Authorship of the Henry VI Plays by Word Adjacency", Shakespeare Quarterly 2016

Authorship Attribution as Graph Signal Processing

F. Gama
fgama@berkeley.edu

- Build WANs that accurately reflect an author's signature requires long texts
  $\Rightarrow$ Attributing short texts becomes challenging even if we have an existing accurate WAN

- The WAN establishes a graph structure $\mathbf{S}$ $\Rightarrow$ Nodes are words, edges are co-occurrence
- Graph signal $\mathbf{x}$ defined on the nodes $\Rightarrow$ Word count of a text $\Rightarrow$ Histogram of function words

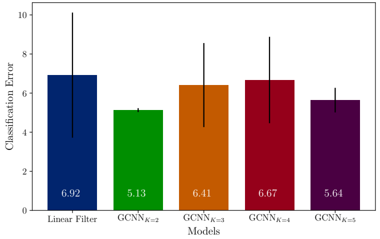
- Use GNNs to classify the authorship of a short text
  $\Rightarrow$ Leverage a given, accurate WAN $\mathbf{S}$ and the word histogram $\mathbf{x}$ of the short text

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2019

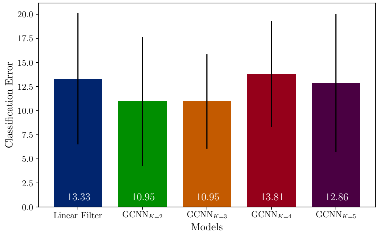
F. Gama
fgama@berkeley.edu

Authorship Attribution: Numerical Results

- Corpus of texts written by Jane Austen and Edgar Allan Poe $\Rightarrow$ Build WAN $\mathbf{S}$
- Pickup pages (1K words) written by J. Austen or E. A. Poe from a pool of 22 contemporaries
  $\Rightarrow$ Compute word count of function words on each page $\Rightarrow$ Graph signals $\mathbf{x}$
- Compare GNNs with different values of K, compare with a linear graph filter



Jane Austen



Edgar Allan Poe

Ruiz, Gama, Marques, Ribeiro, "Invariance-Preserving Localized Activation Functions for Graph Neural Networks ", IEEE TSP 2020

Permutation Equivariance

F. Gama
fgama@berkeley.edu

Graph Neural Networks: Why?

F. Gama
fgama@berkeley.edu

- Time convolutions are intuitive. Graph convolutions not so much.
  $\Rightarrow$ Local information, efficient implementation (distributed)

- When intuition fails we need to resort to first principles $\Rightarrow$ What do we know about CNNs?
- CNNs also exploit the local information and have an efficient implementation
  $\Rightarrow$ Translation equivariance and stability [Mallat '12] $\Rightarrow$ Explain remarkable performance

- Permutation equivariance $\Rightarrow$ Exploit internal symmetries of the graph

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Permutation Equivariance and Node Relabelings

F. Gama
fgama@berkeley.edu

- A permutation $\mathbf{P}$ is a binary matrix that satisfies

$$\{\mathbf{P} \in \{0,1\}^{N \times N} : \mathbf{P}\mathbf{1} = \mathbf{1}, \mathbf{P}^{\mathsf{T}}\mathbf{1} = \mathbf{1}\}$$

- Permuting a vector $\mathbf{P}^{\mathsf{T}}\mathbf{x}$ reorders the entries of the vector
- Permuting a matrix $\mathbf{P}^{\mathsf{T}}\mathbf{S}\mathbf{P}$ reorders the entries of the matrix
- The reordering is correspondent with the linear operation $(\mathbf{P}^{\mathsf{T}}\mathbf{S}\mathbf{P})(\mathbf{P}^{\mathsf{T}}\mathbf{x}) = \mathbf{P}^{\mathsf{T}}(\mathbf{S}\mathbf{x})$

- A permutation amounts to a relabeling of the nodes of the graph
- Choosing a shift operator $\mathbf{S}$ to describe a graph forces a labeling of the nodes
  $\Rightarrow$ This labeling is required for operations. But is arbitrary
- We want signal processing algorithms that are independent of arbitrary labelings

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Permutation Equivariance

- Consider the graph convolution operator $\mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{\infty} h_k \, \mathbf{S}^k \, \mathbf{x}$
- Depends on filter parameters $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$ and shift operator $\mathbf{S}$; applied to the input signal $\mathbf{x}$

Theorem (Gama, Bruna, Ribeiro)

Graph convolutions are equivariant to permutations. For graphs with permuted shift operators $\hat{\mathbf{S}} = \mathbf{P}^{\mathsf{T}}\mathbf{S}\mathbf{P}$ and permuted graph signals $\hat{\mathbf{x}} = \mathbf{P}^{\mathsf{T}}\mathbf{x}$ it holds

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

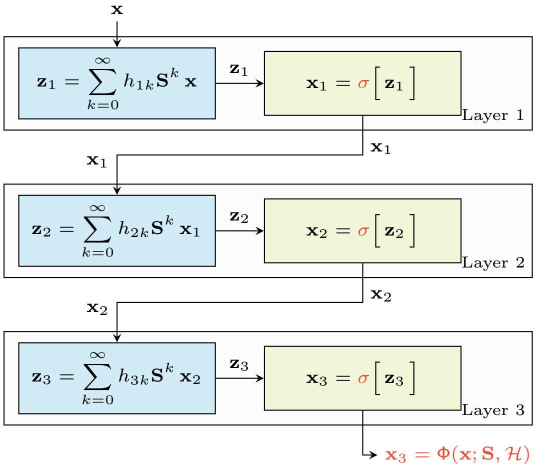
Proof $\Rightarrow$ $\mathbf{H}(\hat{\mathbf{S}})\hat{\mathbf{x}} = \sum_{k=0}^{\infty} h_k \, \hat{\mathbf{S}}^k \hat{\mathbf{x}} = \sum_{k=0}^{\infty} h_k \, (\mathbf{P}^{\mathsf{T}}\mathbf{S}\mathbf{P})^k \mathbf{P}^{\mathsf{T}}\mathbf{x} = \mathbf{P}^{\mathsf{T}} \left(\sum_{k=0}^{\infty} h_k \, \mathbf{S}^k \mathbf{x} \right) = \mathbf{P}^{\mathsf{T}}\mathbf{H}(\mathbf{S})\mathbf{x}$

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

GNNs Inherit Permutation Equivariance from Graph Filters

F. Gama
fgama@berkeley.edu

- GNN is a cascade of layers
  - $\Rightarrow$ Graph filters and pointwise nonlinearities
- A pointwise operation does not mix node values
  - $\Rightarrow$ Independent of the graph
- GNN retains permutation equivariance


$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{\infty} h_{1k}\mathbf{S}^k\,\mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\left[\mathbf{z}_1\right]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{\infty} h_{2k}\mathbf{S}^k\,\mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\left[\mathbf{z}_2\right]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{\infty} h_{3k}\mathbf{S}^k\,\mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\left[\mathbf{z}_3\right]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

Gama, Bruna, Ribeiro, “Stability Properties of Graph Neural Networks”, IEEE TSP 2020.

Permutation Equivariance of Graph Neural Networks

Theorem (Gama, Bruna, Ribeiro)

GNNs are equivariant to permutations. For graphs with permuted shift operators $\hat{\mathbf{S}} = \mathbf{P}^\mathsf{T}\mathbf{S}\mathbf{P}$ *and permuted graph signals* $\hat{\mathbf{x}} = \mathbf{P}^\mathsf{T}\mathbf{x}$ *it holds*

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

where $\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{H})$ *is the output of processing* $\hat{\mathbf{x}}$ *on* $\hat{\mathbf{S}}$ *with GNN* $\mathcal{H}$ *and* $\Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$ *is the output of processing* $\mathbf{x}$ *on* $\mathbf{S}$ *with the same GNN* $\mathcal{H}$*.*

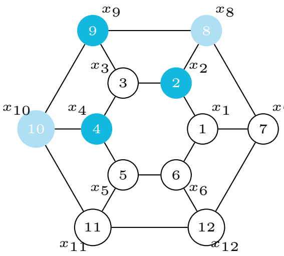
- Signal Processing with Graph Neural Networks is independent of labeling

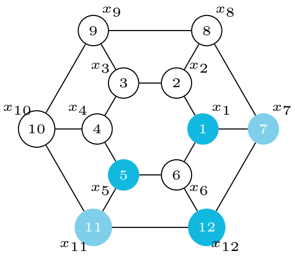
Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Equivariance to Permutations is More Valuable than Apparent

F. Gama
fgama@berkeley.edu

- Invariance to node relabelings allows GNNs to exploit internal symmetries of graph signals


x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}
1 2 3 4 5 6 7 8 9 10 11 12


x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}
1 2 3 4 5 6 7 8 9 10 11 12

- Although different, both signals are permutations of one other
  $\Rightarrow$ Permutation equivariance $\Rightarrow$ The GNN can learn about the right one from seeing the left one

- Permutation Equivariance is not a good idea in all problems $\Rightarrow$ Edge-Variant GNNs

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Isufi, Gama, Ribeiro, "EdgeNets: Edge Varying Graph Neural Networks", `arxiv:2001.07620`, 2020.

Absolute Perturbations

F. Gama
fgama@berkeley.edu

Measuring Distances: Modulo Permutation

- Consider a graph with a shift operator $\mathbf{S}$ and another graph with a shift operator $\hat{\mathbf{S}}$ (same size N)
- Consider a set of filter taps $\{h_k\}$ and filters $\mathbf{H}(\mathbf{S})$ and $\mathbf{H}(\hat{\mathbf{S}})$ with the same filter taps
- We want to characterize how different are $\mathbf{H}(\mathbf{S})$ and $\mathbf{H}(\hat{\mathbf{S}})$

- We know that when $\hat{\mathbf{S}} = \mathbf{P}^\mathsf{T}\mathbf{S}\mathbf{P}$, then $\mathbf{H}(\hat{\mathbf{S}}) = \mathbf{P}^\mathsf{T}\mathbf{H}(\mathbf{S})$ for all permutations $\mathbf{P} \in \mathcal{P}$

  $\Rightarrow$ We want a distance between operators $\mathbf{H}(\mathbf{S})$ and $\mathbf{H}(\hat{\mathbf{S}})$ that is modulo permutations

$$\|\mathbf{H}(\mathbf{S}) - \mathbf{H}(\hat{\mathbf{S}})\|_\mathcal{P} = \min_{\mathbf{P}\in\mathcal{P}} \|\mathbf{H}(\mathbf{P}^\mathsf{T}\mathbf{S}\mathbf{P}) - \mathbf{H}(\hat{\mathbf{S}})\|$$

  $\Rightarrow$ Indeed, if $\hat{\mathbf{S}} = \mathbf{P}^\mathsf{T}\mathbf{S}\mathbf{P}$ then $\|\mathbf{H}(\mathbf{S}) - \mathbf{H}(\hat{\mathbf{S}})\|_\mathcal{P} = 0$ confirming permutation equivariance

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Absolute Perturbations

F. Gama
fgama@berkeley.edu

- We can measure distances between graph filters modulo permutations $\Rightarrow \|\mathbf{H}(\mathbf{S}) - \mathbf{H}(\hat{\mathbf{S}})\|_{\mathcal{P}}$
- We want to characterize this as a function of some distance between $\mathbf{S}$ and $\hat{\mathbf{S}}$ (same filter taps)
  - $\Rightarrow$ We want to know how changing the graph support affects the output of a filter
- Measure the distance between $\mathbf{S}$ and $\hat{\mathbf{S}}$ $\Rightarrow$ Set of absolute error matrices modulo permutations

$$\mathcal{E}(\mathbf{S}, \hat{\mathbf{S}}) = \{\mathbf{E} \in \mathbb{R}^{N \times N} : \mathbf{P}^{\mathsf{T}} \hat{\mathbf{S}} \mathbf{P} = \mathbf{S} + \mathbf{E} \;,\; \mathbf{P} \in \mathcal{P}\}$$

  - $\Rightarrow$ Define absolute perturbation distance as $\mathsf{d}(\mathbf{S}, \hat{\mathbf{S}}) = \min_{\mathbf{E} \in \mathcal{E}(\mathbf{S}, \hat{\mathbf{S}})} \|\mathbf{E}\| \leq \|\hat{\mathbf{S}} - \mathbf{S}\|$
- The effect of the shift operator on the output of a graph filter $\Rightarrow$ Graph frequency domain

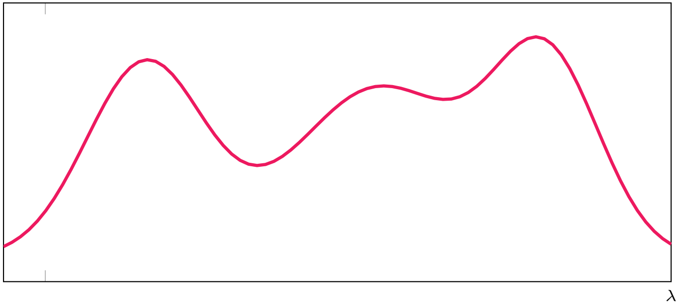
$$\tilde{y}_i = \tilde{\mathsf{h}}(\lambda_i) \tilde{x}_i$$

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Restricting the Class of Filters: Lipschitz Filters

F. Gama
fgama@berkeley.edu

- The frequency response $\tilde{\mathsf{h}}(\lambda)$ of the filter $\mathbf{H}$ satisfies $|\mathsf{h}'(\lambda)| \leq C$



Graph Filter Stability to Absolute Perturbations

F. Gama
fgama@berkeley.edu

Theorem (Gama, Bruna, Ribeiro)

Graph convolutions are stable to absolute perturbations. Given two graphs with shift operators $\mathbf{S}$ *and* $\hat{\mathbf{S}}$ *respectively such that their absolute distances are* $\mathsf{d}(\mathbf{S}, \hat{\mathbf{S}}) \leq \varepsilon$*, then Lipschitz filters with constant* C *satisfy*

$$\|\mathbf{H}(\mathbf{S}) - \mathbf{H}(\hat{\mathbf{S}})\|_{\mathcal{P}} \leq C\ (1 + \delta\sqrt{N})\ \varepsilon + \mathsf{O}(\varepsilon^2)$$

where $\delta = (\|\mathbf{U} - \mathbf{V}\|_2 + 1)^2 - 1 \leq 8$ *is the eigenvector misalignment constant between the eigenvectors* $\mathbf{V}$ *of* $\mathbf{S}$ *and the eigenvectors* $\mathbf{U}$ *of the error matrix* $\mathbf{E}$*.*

- The difference in the output is bounded linearly by the distance between the shift operators
- The bound depends on the filter and on the perturbation model
- The bound is universal for all graphs with the same number of nodes N

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Limitations of the Absolute Perturbation Model

- The theorem is true, and it holds for all graphs. But is not very useful
  - $\Rightarrow$ The absolute perturbation model is too arbitrary $\Rightarrow$ Fails to measure structural graph change
- Absolute perturbations link $\hat{\mathbf{S}}$ with $\mathbf{S}$ through $\mathbf{P}^\mathsf{T}\hat{\mathbf{S}}\mathbf{P} = \mathbf{S} + \mathbf{E}$ $\Rightarrow$ Error matrix $\mathbf{E}$ can be arbitrary

$$\mathbf{P}^\mathsf{T}\begin{bmatrix} & & \\ & \hat{\mathbf{S}} & \\ & & \end{bmatrix}\mathbf{P} = \begin{bmatrix} & & \\ & \mathbf{S} & \\ & & \end{bmatrix} + \begin{bmatrix} & & \\ & \mathbf{E} & \\ & & \end{bmatrix}$$

  - $\Rightarrow$ We can alter the graph topology arbitrarily without changing the value of $\|\mathbf{E}\|$
- We need a better perturbation model $\Rightarrow$ One that takes into account the graph being perturbed

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Relative Perturbations

F. Gama
fgama@berkeley.edu

Relative Perturbations

F. Gama
fgama@berkeley.edu

- We measure distances between graph filters modulo permutations

$$\|\mathbf{H}(\mathbf{S}) - \mathbf{H}(\hat{\mathbf{S}})\|_{\mathcal{P}} = \min_{\mathbf{P}\in\mathcal{P}} \|\mathbf{H}(\mathbf{P}^{\mathsf{T}}\mathbf{S}\mathbf{P}) - \mathbf{H}(\hat{\mathbf{S}})\|$$

- We want to know how changing the graph support affects the output of a filter

- Measure the distance between $\mathbf{S}$ and $\hat{\mathbf{S}}$ $\Rightarrow$ Set of relative error matrices modulo permutations

$$\mathcal{E}(\mathbf{S}, \hat{\mathbf{S}}) = \{\mathbf{E} \in \mathbb{R}^{N\times N} : \mathbf{P}^{\mathsf{T}}\hat{\mathbf{S}}\mathbf{P} = \mathbf{S} + \mathbf{E}^{\mathsf{T}}\mathbf{S} + \mathbf{S}\mathbf{E}\ ,\ \mathbf{P} \in \mathcal{P}\}$$

$\Rightarrow$ Define relative perturbation distance as $\mathsf{d}(\mathbf{S}, \hat{\mathbf{S}}) = \min_{\mathbf{E}\in\mathcal{E}(\mathbf{S},\hat{\mathbf{S}})} \|\mathbf{E}\| \leq \dfrac{\|\hat{\mathbf{S}} - \mathbf{S}\|}{\|\mathbf{S}\|}$

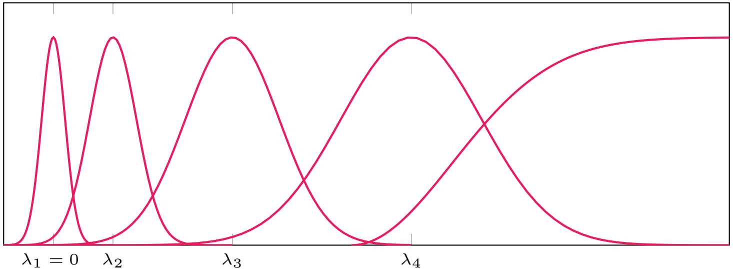
- The effect of the shift operator on the output of a graph filter $\Rightarrow$ Graph frequency domain

$$\tilde{y}_i = \tilde{\mathsf{h}}(\lambda_i)\tilde{x}_i$$

Gama, Bruna, Ribeiro "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Restricting the Class of Filters: Integral Lipschitz Filters

- The frequency response $\mathsf{h}(\lambda)$ of the filter $\mathbf{H}$ satisfies $|\lambda \mathsf{h}'(\lambda)| \leq C$



- Integral Lipschitz filters have to be wide for large λ
- But they can be arbitrarily thin for low λ

Gama, Bruna, Ribeiro "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Graph Filter Stability to Relative Perturbations

F. Gama
fgama@berkeley.edu

Theorem (Gama, Bruna, Ribeiro)

Graph convolutions are stable to relative perturbations. Given two graphs with shift operators $\mathbf{S}$ *and* $\hat{\mathbf{S}}$ *respectively such that their relative distances are* $\mathsf{d}(\mathbf{S}, \hat{\mathbf{S}}) \leq \varepsilon$*, then integral Lipschitz filters with constant* C *satisfy*

$$\|\mathbf{H}(\mathbf{S}) - \mathbf{H}(\hat{\mathbf{S}})\|_{\mathcal{P}} \leq 2C \ (1 + \delta\sqrt{N}) \ \varepsilon + \mathrm{O}(\varepsilon^2)$$

where $\delta = (\|\mathbf{U} - \mathbf{V}\|_2 + 1)^2 - 1 \leq 8$ *is the eigenvector misalignment constant between the eigenvectors* $\mathbf{V}$ *of* $\mathbf{S}$ *and the eigenvectors* $\mathbf{U}$ *of the error matrix* $\mathbf{E}$*.*

- The difference in the output is bounded linearly by the distance between the shift operators
- The bound depends on the filter and on the perturbation model
- The bound is universal for all graphs with the same number of nodes N

GNN Stability to Relative Perturbations

F. Gama
fgama@berkeley.edu

Theorem (Gama, Bruna, Ribeiro)

Graph neural networks are stable to relative perturbations. Let $\mathbf{S}$ *and* $\hat{\mathbf{S}}$ *be the corresponding shift operators of two graphs such that their relative distances are* $\mathsf{d}(\mathbf{S}, \hat{\mathbf{S}}) \leq \varepsilon$. *Let* $\Phi(\cdot; \mathbf{S}, \mathcal{H})$ *be a GNN with* L *layers and* F *features per layer and with integral Lipschiz filters with constant* C. *Then, it holds that*

$$\|\Phi(\cdot; \mathbf{S}, \mathcal{H}) - \Phi(\cdot; \hat{\mathbf{S}}, \mathcal{H})\|_{\mathcal{P}} \leq 2C \ (1 + \delta\sqrt{N}) \ LF^{L-1} \ \varepsilon + \mathrm{O}(\varepsilon^2)$$

where $\delta = (\|\mathbf{U} - \mathbf{V}\|_2 + 1)^2 - 1 \leq 8$ *is the eigenvector misalignment constant between the eigenvectors* $\mathbf{V}$ *of* $\mathbf{S}$ *and the eigenvectors* $\mathbf{U}$ *of the error matrix* $\mathbf{E}$.

- The difference in the output is bounded linearly by the distance between the shift operators
- The bound depends on the filter, the architecture design and on the perturbation model
- The bound is universal for all graphs with the same number of nodes N

Gama, Bruna, Ribeiro “Stability Properties of Graph Neural Networks”, IEEE TSP 2020.

Stability Insights

F. Gama
fgama@berkeley.edu

Graph Filters or GNNs?

F. Gama
fgama@berkeley.edu

- Permutation equivariance and stability are properties of both graph convolutions and GNNs
- Why choose GNNs over graph convolutions?
  - ⇒ Q1: What is good about pointwise nonlinearities?
  - ⇒ Q2: What is wrong with linear graph convolutions?
- A2: They can be unstable to perturbations of the graph if we push their discriminative power
- A1: They make GNNs stable to perturbations while retaining discriminability
- These questions can be answered with an analysis in the **spectral domain**

$$\tilde{y}_i = \tilde{\mathsf{h}}(\lambda_i)\tilde{x}_i$$

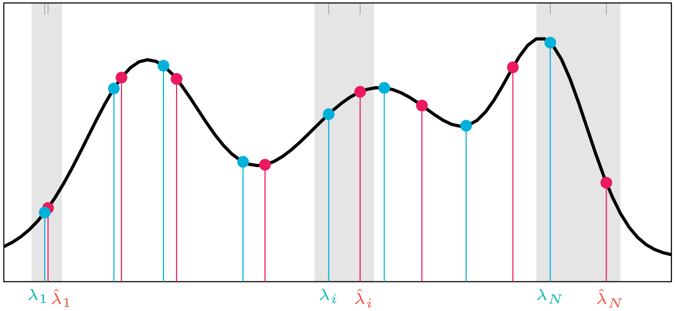
Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Proof of Stability Theorem

F. Gama
fgama@berkeley.edu

- The GNN stability theorem is elementary to prove for an edge dilation $\Rightarrow \hat{\mathbf{S}} = (1+\varepsilon)\mathbf{S}$

- An edge dilation just produces a spectrum dilation $\Rightarrow \hat{\lambda}_i = (1+\varepsilon)\lambda_i$, $\mathbf{E} = (\varepsilon/2)\mathbf{I}$



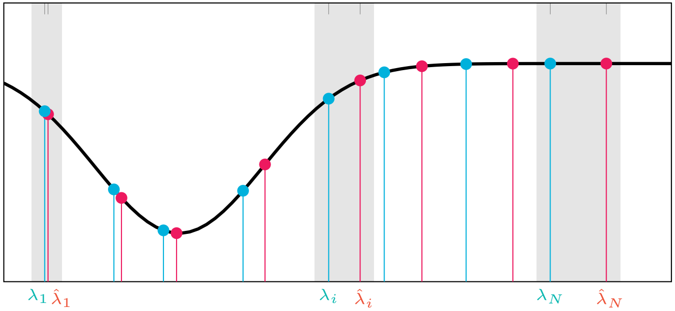
- Small deformations may result in large filter variations for large λ if filter is not integral Lipschitz

Gama, Bruna, Ribeiro, “Stability Properties of Graph Neural Networks”, IEEE TSP 2020.

Proof of Stability Theorem

F. Gama
fgama@berkeley.edu

- The GNN stability theorem is elementary to prove for an edge dilation $\Rightarrow \hat{\mathbf{S}} = (1+\varepsilon)\mathbf{S}$
- An edge dilation just produces a spectrum dilation $\Rightarrow \hat{\lambda}_i = (1+\varepsilon)\lambda_i$, $\mathbf{E} = (\varepsilon/2)\mathbf{I}$



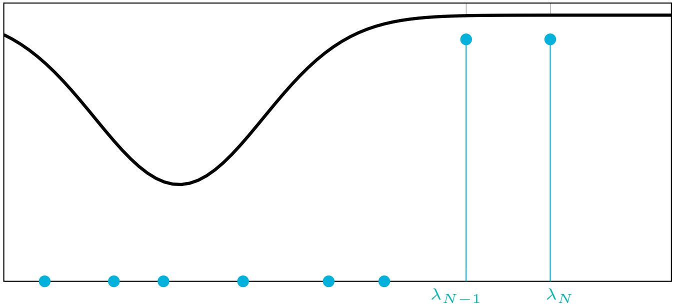
- Integral Lipschitz is always stable $\Rightarrow$ Eigenvalue does not move or filter does not move

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Discriminative Graph Filter Banks are Unstable

F. Gama
fgama@berkeley.edu

- Q2: What is wrong with linear graph convolutions?
- Cannot be simultaneously stable to deformations and discriminate features at large eigenvalues



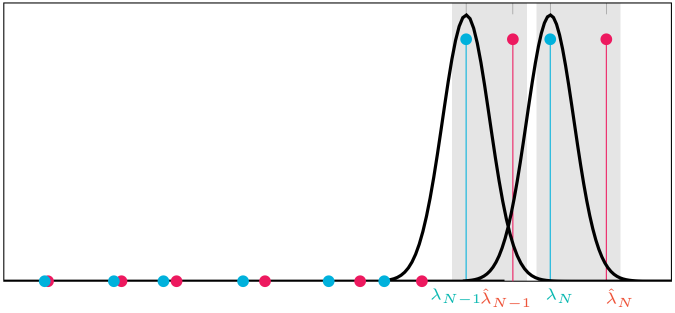
- Limits their value in machine learning problems where features at large eigenvalues are important

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Discriminative Graph Filter Banks are Unstable

F. Gama
fgama@berkeley.edu

- Q2: What is wrong with linear graph convolutions?
- Cannot be simultaneously stable to deformations and discriminate features at large eigenvalues



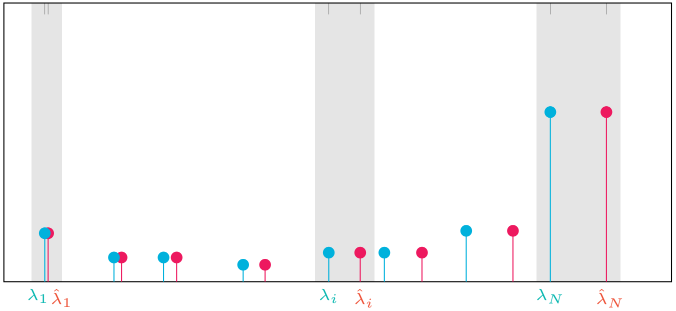
- Limits their value in machine learning problems where features at large eigenvalues are important

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Nonlinearities Create Low Frequency Components

F. Gama
fgama@berkeley.edu

- Q1: What is good about pointwise nonlinearities?
- Preserve permutation equivariance while generating low graph frequency components
  $\Rightarrow$ Which we can discriminate with stable filters



Spectrum of rectified graph signal

$$\mathbf{x}_{\text{relu}} = \max(\mathbf{x}, 0)$$

- The nonlinearity demodulates. It creates low frequency content that is stable

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Nonlinearities Create Low Frequency Components

F. Gama
fgama@berkeley.edu

- Q1: What is good about pointwise nonlinearities?
- Preserve permutation equivariance while generating low graph frequency components
  $\Rightarrow$ Which we can discriminate with stable filters

GNNs are **stable** and **selective** information processing architectures

- The nonlinearity demodulates. It creates low frequency content that is stable

Gama, Bruna, Ribeiro, “Stability Properties of Graph Neural Networks”, IEEE TSP 2020.

Learning Decentralized Control

F. Gama
fgama@berkeley.edu

Decentralized Coordination of a Robot Swarm

- We want the team to coordinate on their individual velocities without colliding with each other

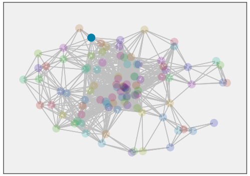
- This is a very easy problem to solve if we allow for centralized coordination $\Rightarrow \mathbf{u}_i = \sum_{i=1}^{N} \mathbf{v}_i$
- But it is very difficult to solve if we do do not allow for centralized coordination $\Rightarrow \mathbf{u}_i = \ldots$

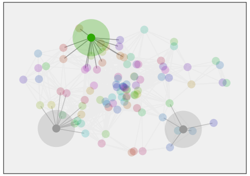
Tolstaya, Gama, Paulos, Pappas, Kumar, Ribeiro, "Learning Decentralized Controllers for Robot Swarms with GNNs", CoRL 2019.

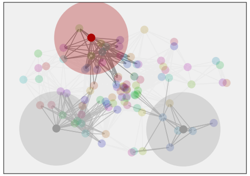
Information Structure on Distributed Systems

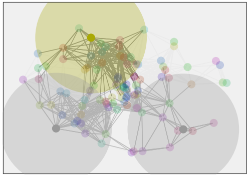
F. Gama
fgama@berkeley.edu

- The challenge in designing behaviors for distributed systems is the partial information structure


$\mathbf{x}_i(t)$


$\mathbf{x}_j(t-1)$ for $j \in \mathcal{N}_i^1$


$\mathbf{x}_j(t-2)$ for $j \in \mathcal{N}_i^2$


$\mathbf{x}_j(t-3)$ for $j \in \mathcal{N}_i^3$

- Node i has access to its own local information at time t $\Rightarrow \mathbf{x}_i(t)$
- And the information of its 1-hop neighbors at time $t-1$ $\Rightarrow \mathbf{x}_j(t-1)$ for all $j \in \mathcal{N}_i^1$
- And the information of its 2-hop neighbors at time $t-2$ $\Rightarrow \mathbf{x}_j(t-2)$ for all $j \in \mathcal{N}_i^2$
- And the information of its 3-hop neighbors at time $t-3$ $\Rightarrow \mathbf{x}_j(t-3)$ for all $j \in \mathcal{N}_i^3$

Tolstaya, Gama, Paulos, Pappas, Kumar, Ribeiro, "Learning Decentralized Controllers for Robot Swarms with GNNs", CoRL 2019.

Learning Distributed Actions

F. Gama
fgama@berkeley.edu

- Optimal centralized actions act directly on all states $\pi^*(\mathbf{x}(t))$ and can be readily computed

- Distributed actions can only depend on information history $\Rightarrow \mathcal{X}_i(t) = \bigcup_{k=0}^{K-1} \left\{ \mathbf{x}_j(t-k) : j \in \mathcal{N}_i^k \right\}$

  $\Rightarrow$ Optimal distributed actions $\pi^*(\mathcal{X}_i(t))$ are famously difficult to find [Witsenhausen '68]

  $\Rightarrow$ When optimal solutions are out of reach we resort to heuristics $\Rightarrow$ data driven heuristics

Gama, Li, Tolstaya, Prorok, Ribeiro, "Decentralized Control with Graph Neural Networks", arxiv.org/abs/2012.14906

Imitation Learning of Distributed Actions

- Parametrize $\pi(\mathcal{X}_i(t))$ with a graph neural network $\pi(\mathcal{X}_i(t), \mathcal{H})$
  - $\Rightarrow$ Adopt learning architecture that naturally processes distributed partial information $\mathcal{X}_i(t)$
- Train $\pi(\mathcal{X}_i(t), \mathcal{H})$ by using **imitation learning**
- Optimal centralized policy $\pi^*(\mathbf{x}(t))$ can be computed during training time
- Find the parameters $\mathcal{H}$ that make $\pi(\mathcal{X}_i(t), \mathcal{H})$ closer to $\pi^*(\mathbf{x}(t))$

$$\mathcal{H}^* = \arg\min_{\mathcal{H}} \mathbb{E}_{\pi^*}\left[\mathsf{J}\Big(\pi\big(\mathcal{X}_i(t), \mathcal{H}\big), \pi^*\big(\mathbf{x}(t)\big)\Big)\right]$$

- Centralized policy required at train time but **not at test time**

Gama, Li, Tolstaya, Prorok, Ribeiro, "Decentralized Control with Graph Neural Networks", `arxiv.org/abs/2012.14906`

GNN Properties Play a Key Role

F. Gama
fgama@berkeley.edu

- GNN policies $\pi(\mathcal{X}_i(t), \mathcal{H}) = \Phi(\mathbf{x}(t); \mathbf{S}(t), \mathcal{H})$ respect the partial information structure $\mathcal{X}_i(t)$
  - $\Rightarrow$ Graph filters $\mathbf{H}(\mathcal{X}_i(t)) = \sum_{k=0}^{K-1} h_k \, \mathbf{S}(t) \cdots \mathbf{S}(t-(k-1)) \mathbf{x}(t-k)$
- Permutation equivariance
  - $\Rightarrow$ If two agents observe the same input
  - $\Rightarrow$ Their k-hop neighbors observe the same inputs
  - $\Rightarrow$ And the local neighborhood structures of the graph are the same
- Then the output of the control policy is the same at both nodes
- Stability $\Rightarrow$ If graphs are similar, GNN outputs will be similar
- These properties are a necessity for offline training

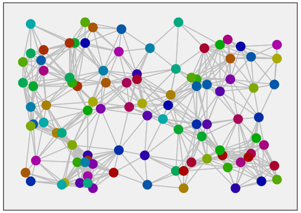
Gama, Li, Tolstaya, Prorok, Ribeiro, "Decentralized Control with Graph Neural Networks", `arxiv.org/abs/2012.14906`

Offline Training vs Online Execution

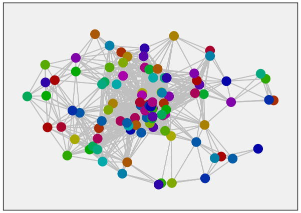
F. Gama
fgama@berkeley.edu

- If we want to train offline and execute online we can't assume the graph is the same

- Train offline on a graph like this



- And execute online on a graph like this



- Graph convolutions $\Rightarrow$ Learn parameters $\mathcal{H}$ $\Rightarrow$ Run on any $\mathbf{S}$: $\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{K-1} h_k \, \mathbf{S}^k$
- Permutation equivariance $\Rightarrow$ Exploit symmetries, reuse data
- Stability $\Rightarrow$ Similar graphs yield similar outputs

Gama, Li, Tolstaya, Prorok, Ribeiro, "Decentralized Control with Graph Neural Networks", arxiv.org/abs/2012.14906

Flocking

Team Dynamics

F. Gama
fgama@berkeley.edu

- Team of N agents with positions $\mathbf{r}_i(t)$, velocities $\mathbf{v}_i(t)$ and acceleration $\mathbf{u}_i(t)$
- System dynamics $\Rightarrow$ Constant acceleration during each interval T_s

$$\begin{cases} \mathbf{r}_i(t+1) = \mathbf{u}_i(t)T_s^2/2 + \mathbf{v}_i(t)T_s + \mathbf{r}_i(t) \\ \mathbf{v}_i(t+1) = \mathbf{u}_i(t)T_s + \mathbf{v}_i(t) \end{cases}$$

- Control acceleration $\mathbf{u}_i(t)$ $\Rightarrow$ Design sequence of acceleration values $\{\mathbf{u}_i(t)\}$
- Proof of concept $\Rightarrow$ More realistic scenarios are available [Tolstaya et al '19, Hu et al '20, Li et al '20]

Tolstaya, Gama, Paulos, Pappas, Kumar, Ribeiro, "Learning Decentralized Controllers for Robot Swarms with GNNs", CoRL 2019.

Li, Gama, Ribeiro, Prorok, "Graph Neural Networks for Decentralized Multi-Robot Path Planning", IRoS 2020.

Hu, Gama, Wang, Ribeiro, Sadler "VGAI: A Vision-Based Decentralized Controller Learning Framework for Robot Swarms", `arxiv.org/abs/2002.02308`

Gama, Li, Tolstaya, Prorok, Ribeiro, "Decentralized Control with Graph Neural Networks", `arxiv.org/abs/2012.14906`

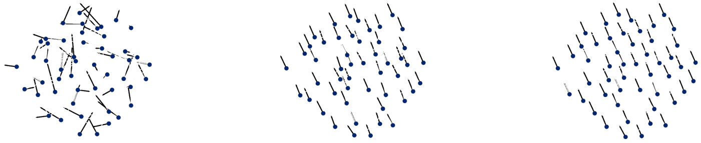
Problem Statement

F. Gama
fgama@berkeley.edu

- Coordinate the velocities $\mathbf{v}_i(t)$ of all agents to be the same while avoiding collisions
- Measure the cost by computing the velocity variance of the team for the entire trajectory

$$\mathsf{J}\big[\{\mathbf{v}_i(t)\}\big] = \frac{1}{N}\sum_t \sum_{i=1}^{N} \|\mathbf{v}_i(t) - \bar{\mathbf{v}}_j(t)\|^2 \ , \ \bar{\mathbf{v}}_j = \frac{1}{N}\sum_{j=1}^{N} \mathbf{v}_j(t)$$

- Control accelerations $\mathbf{u}_i(t)$ such that $\min_{\mathbf{u}_i(t)\ ,\ t\geq 0} \mathsf{J}\big[\{\mathbf{v}_i(t)\}\big] \Rightarrow$ Velocity $\mathbf{v}_i(t+1) = \mathbf{u}_i(t)T_s + \mathbf{v}_i(t)$

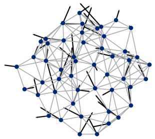


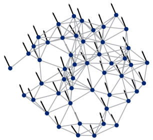
t = 0s t = 1s t = 2s

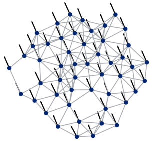
Communication Network

- Decentralized setting $\Rightarrow$ Agents can only communicate if $\|\mathbf{r}_i(t) - \mathbf{r}_j(t)\| \leq R$
- Communication graph $\mathbf{S}(t)$ $\Rightarrow$ Edge $[\mathbf{S}(t)]_{ij}$ if agents can communicate
- Use GNNs that respect the communicaton graph $\Rightarrow$ Local and distributed operations

$$\mathbf{U}(\mathcal{X}_i(t), \mathcal{H}) = \mathbf{\Phi}(\mathbf{X}(t); \mathbf{S}(t), \mathcal{H}) \ , \ \mathbf{H}(\mathcal{X}_i(t)) = \sum_{k=0}^{K-1} h_k \ \mathbf{S}(t) \cdots \mathbf{S}(t-(k-1))\mathbf{x}(t-k)$$


$t = 0$s


$t = 1$s


$t = 2$s

Imitation Learning

F. Gama
fgama@berkeley.edu

- Train GNN $\mathbf{\Phi}(\mathbf{X}(t); \mathbf{S}(t), \mathcal{H})$ by using imitation learning $\Rightarrow$ Find $\mathcal{H}$ that make $\mathbf{\Phi}$ closer to $\mathbf{U}^\star$

$$\mathcal{H}^\star = \arg\min_{\mathcal{H}} \mathbb{E}_{\mathbf{U}^\star}\Big[\|\mathbf{\Phi}(\mathbf{X}(t); \mathbf{S}(t), \mathcal{H}) - \mathbf{U}^\star\|\Big]$$

- The optimal centralized solution $\mathbf{U}^\star$ that avoids collisions is given by

$$\mathbf{u}_i^\star(t) = -\sum_{j=1}^{N}\big(\mathbf{v}_i(t) - \mathbf{v}_j(t)\big) \underbrace{-\sum_{j=1}^{N} \nabla_{\mathbf{r}_i(t)} P(\mathbf{r}_i(t), \mathbf{r}_j(t))}_{\text{collision avoidance}}$$

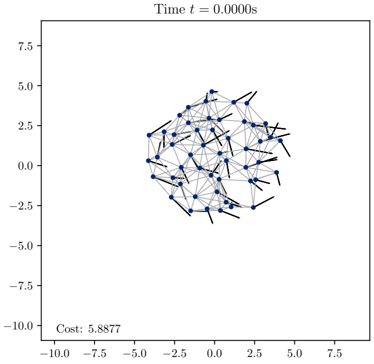
- The state $\mathbf{x}_i(t)$ for each agent is set to $\nabla_{\mathbf{r}_i(t)} P(\mathbf{r}_i(t), \mathbf{r}_j(t))$ $\Rightarrow$ Local to each agent
- Centralized controller $\mathbf{U}^\star$ is required at train time **but not** at testing time

Gama, Li, Tolstaya, Prorok, Ribeiro, "Decentralized Control with Graph Neural Networks", arxiv.org/abs/2012.14906

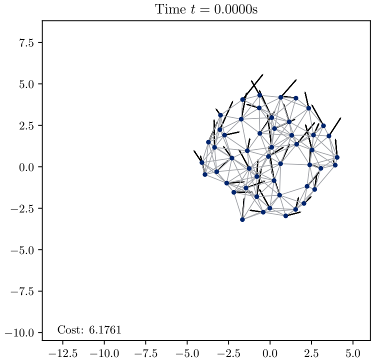
Flocking: Transferring

F. Gama
fgama@berkeley.edu

- 50 Agents.



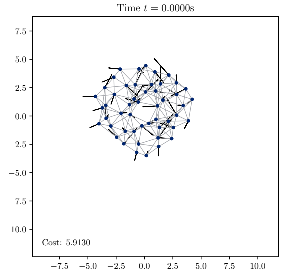
Offline optimal trajectory



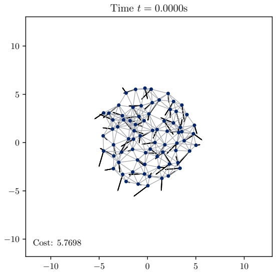
Online learned trajectory

Flocking: Scalability

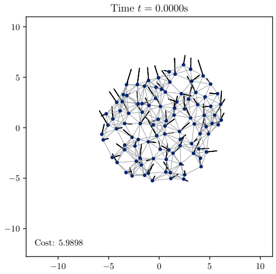
F. Gama
fgama@berkeley.edu



50 agents.



75 agents.



100 agents.

Flocking: Transferring at Scale

F. Gama
fgama@berkeley.edu

- Train on 50 agents. Test on 100 agents.

Flocking: Takeaways

- GNNs learn controllers to drive a team of agents to fly at the same velocity while avoiding collisions
  $\Rightarrow$ Control actions taken by each agent based only on outdated neighboring information
- Transfer $\Rightarrow$ The communication network changes from time to time $\Rightarrow$ GNNs work
- Scale $\Rightarrow$ Learn to control teams of increasing number of agents $\Rightarrow$ GNNs work
- Transfer at scale $\Rightarrow$ Train on a small team, test on a big team $\Rightarrow$ GNNs work

Tolstaya, Gama, Paulos, Pappas, Kumar, Ribeiro, "Learning Decentralized Controllers for Robot Swarms with GNNs", CoRL 2019.

Gama, Li, Tolstaya, Prorok, Ribeiro, "Decentralized Control with Graph Neural Networks", `arxiv.org/abs/2012.14906`

Conclusions

F. Gama
fgama@berkeley.edu

Quiz

F. Gama
fgama@berkeley.edu

What?	$\Rightarrow$	Machine learning on graphs
Why?	$\Rightarrow$	Generic models of signal structure.
		Models of physical infrastructure

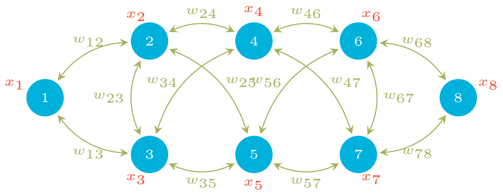
How?	$\Rightarrow$	Graph Neural Networks
Does this work?	$\Rightarrow$	Yes: Examples
Should this work?	$\Rightarrow$	Yes: Equivariance and Stability

How? Signals Supported on Graphs

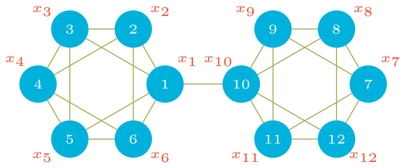
F. Gama
fgama@berkeley.edu

- We saw several systems (and there are many more) that can be modeled with graph signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph



Another signal supported on another graph



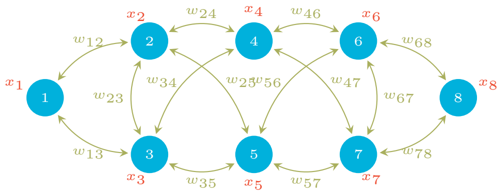
- Nodes are words. Signal values are word frequencies. Edges are word co-occurrence frequencies

How? Signals Supported on Graphs

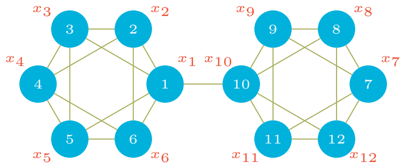
F. Gama
fgama@berkeley.edu

- We saw several systems (and there are many more) that can be modeled with graph signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph



Another signal supported on another graph



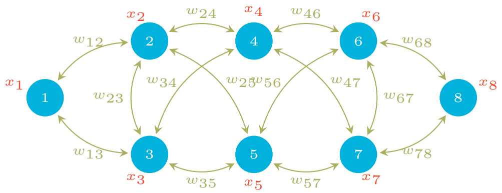
- Nodes are products. Signal values are product ratings. Edges are cosine similarities of past scores

How? Signals Supported on Graphs

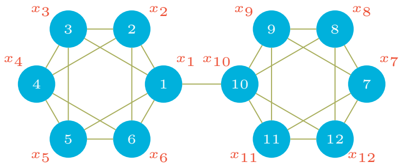
F. Gama
fgama@berkeley.edu

- We saw several systems (and there are many more) that can be modeled with graph signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph



Another signal supported on another graph



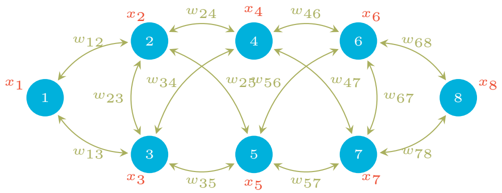
- Nodes are drones. Signal values are velocities. Edges are sensing and communication ranges

How? Signals Supported on Graphs

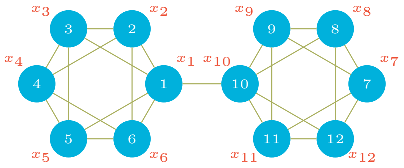
F. Gama
fgama@berkeley.edu

- We saw several systems (and there are many more) that can be modeled with graph signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph



Another signal supported on another graph

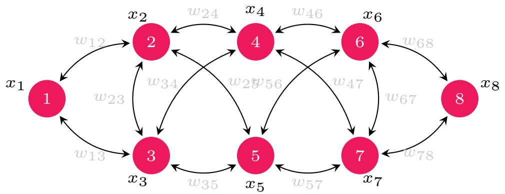


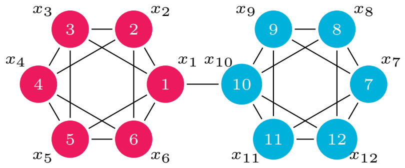
- Nodes are transceivers. Signal values are QoS requirements. Edges are wireless channels strength

How? Graph Filters $\equiv$ Graph Convolutions

F. Gama
fgama@berkeley.edu

- Graph filter on shift operator $\mathbf{S}$ with coefficients h_k is a polynomial on $\mathbf{S}$ $\Rightarrow \mathbf{H}(\mathbf{S}) = \sum_{k=0}^{\infty} h_k \mathbf{S}^k$
- A graph convolution is the result of applying a graph filter $\Rightarrow \mathbf{z} = \mathbf{h} *_{\mathbf{S}} \mathbf{x} = \mathbf{H}(\mathbf{S})\mathbf{x}$





- Shares/Inherits locality of time convolutions $\Rightarrow \mathbf{z} = h_0\mathbf{S}^0\mathbf{x} + h_1\mathbf{S}^1\mathbf{x} + h_2\mathbf{S}^2\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\mathbf{S}^k\mathbf{x}$
- The output of a graph filter depends on the filter coefficients $\mathbf{h}$ and the graph shift operator $\mathbf{S}$

How? Graph Neural Networks (GNNs)

F. Gama
fgama@berkeley.edu

- A graph perceptron is the compositions of a graph filter with a pointwise nonlinearity

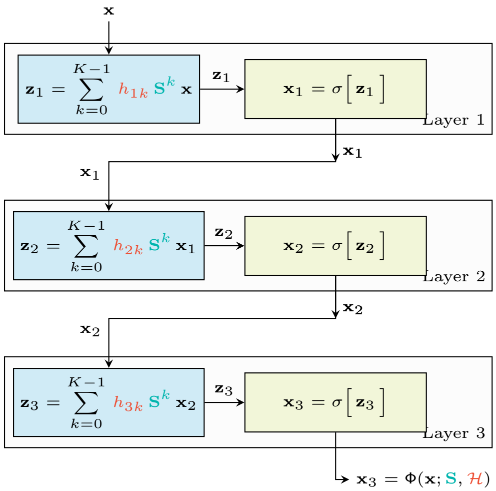
- Layer a few graph perceptrons (3 in the figure)

  $\Rightarrow$ Feed the input signal $\mathbf{x}$ to Layer 1

  $\Rightarrow$ Connect output of Layer 1 to input of Layer 2

  $\Rightarrow$ And output of Layer 2 to input of Layer 3

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

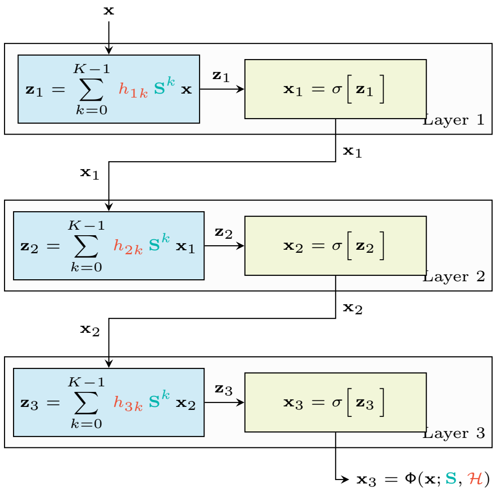

$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k} \mathbf{S}^k \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\left[\mathbf{z}_1\right]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k} \mathbf{S}^k \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\left[\mathbf{z}_2\right]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k} \mathbf{S}^k \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\left[\mathbf{z}_3\right]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

Gama, Marques, Leus, Ribeiro, “Convolutional Neural Network Architectures for Signals Supported on Graphs”, IEEE TSP 2019.

How? Graph Neural Networks (GNNs)

F. Gama
fgama@berkeley.edu

- A GNN is a minor variation of a graph filter
  - $\Rightarrow$ Pointwise nonlinearities and compositions
- GNNs can be transferred across different graphs
  - $\Rightarrow$ Graph $\mathbf{S}$ reinterpreted as an input
- Convolutional (C)NNs are a particular case
  - $\Rightarrow$ When $\mathbf{S}$ is adjacency of a line graph

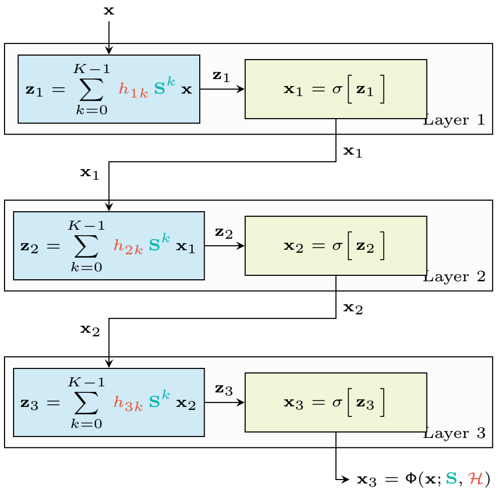

$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k} \mathbf{S}^k \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\left[\mathbf{z}_1\right]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k} \mathbf{S}^k \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\left[\mathbf{z}_2\right]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k} \mathbf{S}^k \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\left[\mathbf{z}_3\right]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2019.

F. Gama
fgama@berkeley.edu

How? Graph Neural Networks (GNNs)

- Basic architecture $\Rightarrow$ A lot more goes on in practice
  - $\Rightarrow$ Multiple features.
  - $\Rightarrow$ Pooling.
- Including many other things we did not cover
  - $\Rightarrow$ Local Nonlinear Activation.
  - $\Rightarrow$ Edge Varying Graph Filters.
  - $\Rightarrow$ Recurrent GNNs and Gating

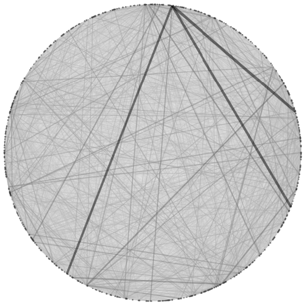
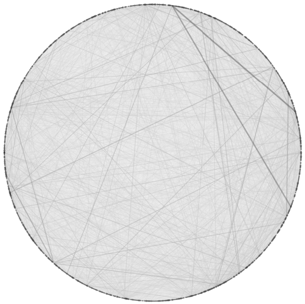

$\mathbf{x}$
$\mathbf{z}_1 = \sum_{k=0}^{K-1} h_{1k} \mathbf{S}^k \mathbf{x}$
$\mathbf{z}_1$
$\mathbf{x}_1 = \sigma\left[\mathbf{z}_1\right]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{z}_2 = \sum_{k=0}^{K-1} h_{2k} \mathbf{S}^k \mathbf{x}_1$
$\mathbf{z}_2$
$\mathbf{x}_2 = \sigma\left[\mathbf{z}_2\right]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{z}_3 = \sum_{k=0}^{K-1} h_{3k} \mathbf{S}^k \mathbf{x}_2$
$\mathbf{z}_3$
$\mathbf{x}_3 = \sigma\left[\mathbf{z}_3\right]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$

Gama, Marques, Leus, Ribeiro, “Convolutional Neural Network Architectures for Signals Supported on Graphs”, IEEE TSP 2019.

Does this work? Recommendation Systems

F. Gama
fgama@berkeley.edu

- GNNs fill ratings with state of the art accuracy $\Rightarrow$ Exploit similarities between past ratings
- Leveraging structure is necessary for scalable learning. There's no such thing as model free learning

Ruiz, Gama, Ribeiro, "Invariance-Preserving Localized Activation Functions for Graph Neural Networks", IEEE TSP 2019.

Does this work? Distributed Collaborative Intelligent Systems

- Graphs are also intrinsic to physical systems $\Rightarrow$ Where they are the source of the problem
  $\Rightarrow$ GNNs can be implemented in a distributed manner $\Rightarrow$ Flock as well as centralized controller

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

Does this work? Distributed Collaborative Intelligent Systems

- GNNs have great potential for data-driven (simulation driven) optimization in DCIS
  - $\Rightarrow$ Because they leverage underlying graphs and can be implemented in a distributed manner
  - $\Rightarrow$ Formation control $\Rightarrow$ Maintain formation without colliding and tolerating wind disturbances

Khan, Tolstaya, Ribeiro, Kumar, "Graph Policy Gradients for Large Scale Robot Control, `arxiv.org/abs/1907.03822`

Does this work? Distributed Collaborative Intelligent Systems

- GNNs have great potential for data-driven (simulation driven) optimization in DCIS
  - ⇒ Because they leverage underlying graphs and can be implemented in a distributed manner
  - ⇒ Flocking with visual inputs ⇒ Use a CNN to produce features that feed a GNN

Hu, Gama, Wang, Ribeiro, Sadler, "VGAI: A Vision-Based Decentralized Controller Learning Framework for Robot Swarms, `arxiv.org/abs/2002.02308`

Does this work? Distributed Collaborative Intelligent Systems

F. Gama
fgama@berkeley.edu

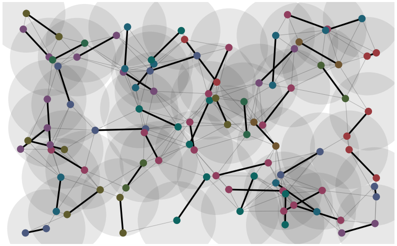
- GNNs have great potential for data-driven (simulation driven) optimization in DCIS
  - $\Rightarrow$ Because they leverage underlying graphs and can be implemented in a distributed manner
  - $\Rightarrow$ Multi Robot Navigation $\Rightarrow$ n robots have to reach n goals in cluttered environment

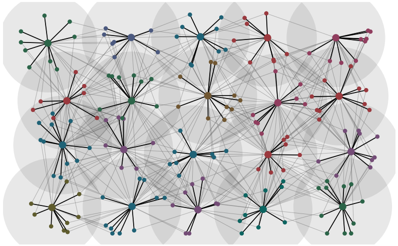
Li, Gama, Ribeiro, Prorok, "Graph Neural Networks for Decentralized Multi-Robot Path Planning", IRoS 2020.

F. Gama
fgama@berkeley.edu

Does this work? GNNs in Large Scale Infrastructure

- GNNs have great potential for data-driven (simulation driven) optimization in decentralized systems
  - ⇒ Because they leverage underlying graphs and can be implemented in a distributed manner
  - ⇒ Wireless Networks ⇒ Allocate resources over time varying channels



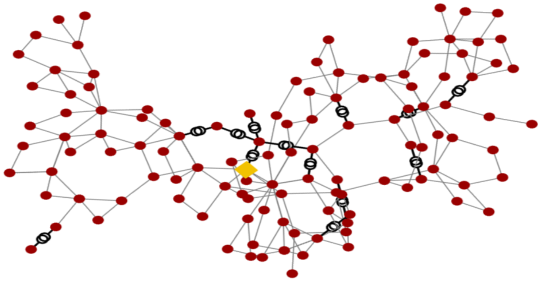


Eisen, Ribeiro, "Optimal Wireless Resource Allocation with Random Edge Graph Neural Networks", IEEE TSP 2020.

Does this work? GNNs in Large Scale Infrastructure

F. Gama
fgama@berkeley.edu

- GNNs have great potential for data-driven (simulation driven) optimization in decentralized systems
  - $\Rightarrow$ Because they leverage underlying graphs and can be implemented in a distributed manner
  - $\Rightarrow$ Power Distribution Grids $\Rightarrow$ (Approximate) Decentralized solution of optimal power flow (OPF)



Owerko, Gama, Ribeiro, “Optimal Power Flow Using Graph Neural Networks”, IEEE ICASSP 2020.

Should this work? Permutation Equivariance of GNNs

F. Gama
fgama@berkeley.edu

- GNNs are equivariant to permutations.
  - $\Rightarrow$ Signal Processing with Graph Neural Networks is independent of labeling

Theorem (Gama, Bruna, Ribeiro)

We are given graph signal $(\mathbf{S}, \mathbf{x})$ *and apermuted graph signal* $(\hat{\mathbf{S}}, \hat{\mathbf{x}})$ *with* $\hat{\mathbf{S}} = \mathbf{P}^\top \mathbf{S} \mathbf{P}$ *and* $\hat{\mathbf{x}} = \mathbf{P}^\top \mathbf{x}$.

We run the same GNN tensor $\mathcal{H}$ *on both producing outputs* $\Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$ *and* $\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{H})$.

The outputs are related by the respective permutation

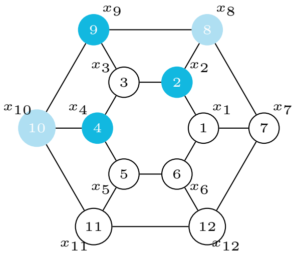
$$\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{H}) \;=\; \mathbf{P}^\top \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$$

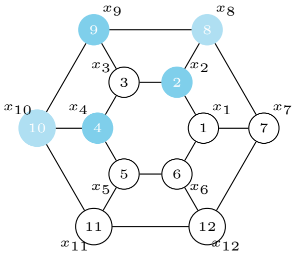
Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

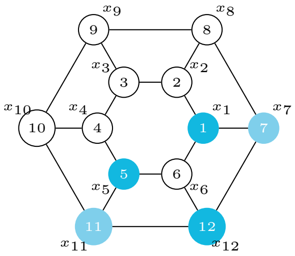
Should this work? Permutation Equivariance of GNNs

F. Gama
fgama@berkeley.edu

- Permutation equivariance explains the ability of GNNS to leverage signal structure for generalization
  - ⇒ Neural Networks, fully connected or graph, generalize from the left to the middle
  - ⇒ Graph Neural Networks generalize from the left to the right







Should this work? Stability to Perturbations of the Graph

F. Gama
fgama@berkeley.edu

- GNNs can be made uniformly Lipschitz stable with respect to relative perturbations of graphs

Theorem (Gama, Bruna, Ribeiro)

Let $\mathbf{S}$ *and* $\hat{\mathbf{S}}$ *be shift operators of graphs with relative distance* $d(\mathbf{S}, \hat{\mathbf{S}}) \leq \varepsilon$. *Let* $\mathbf{\Phi}(\cdot; \mathbf{S}, \mathcal{H})$ *be a GNN with* L *layers and* F *features per layer and with integral Lipschiz filters with constant* C. *Then,*

$$\left\| \mathbf{\Phi}(\cdot; \mathbf{S}, \mathcal{H}) - \mathbf{\Phi}(\cdot; \hat{\mathbf{S}}, \mathcal{H}) \right\|_{\mathcal{P}} \leq 2C \left(1 + \delta\sqrt{N}\right) LF^{L-1} \varepsilon + \mathcal{O}(\varepsilon^2)$$

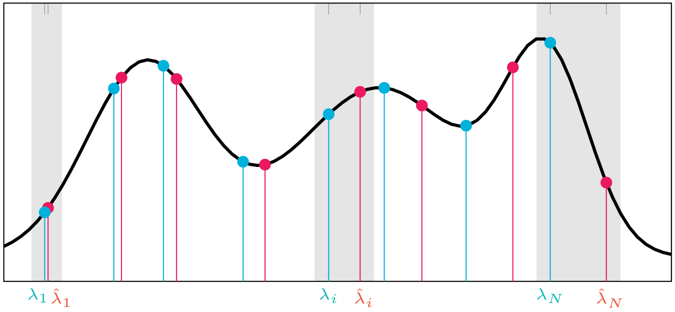
where $\delta = (\|\mathbf{U} - \mathbf{V}\|_2 + 1)^2 - 1 \leq 8$ *is the eigenvector misalignment constant between the eigenvectors* $\mathbf{V}$ *of the graph*$\mathbf{S}$ *and the eigenvectors* $\mathbf{U}$ *of the error matrix* $\mathbf{E}$.

Gama, Bruna, Ribeiro, "Stability Properties of Graph Neural Networks", IEEE TSP 2020.

Should this work? Stability to Perturbations of the Graph

F. Gama
fgama@berkeley.edu

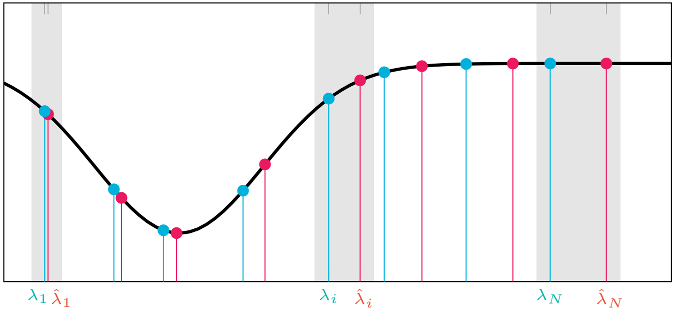
- Given coefficients $\mathbf{h} = \{h_k\}_{k=1}^{\infty}$ the graph filter frequency response is determined $\tilde{\mathsf{h}}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$
- For a given graph $\mathbf{S}$, the response is instantiated on its specific eigenvalues λ_i
- For a dilated graph $\mathbf{S} = (1+\varepsilon)\mathbf{S}$, the response is instantiated at dilated eigenvalues $\hat{\lambda}_i = (1+\varepsilon)\lambda_i$
- Can't have universal Lipschitz constant $\Rightarrow$ arbitrarily large eigenvalues can move arbitrarily much



Should this work? Stability to Perturbations of the Graph

F. Gama
fgama@berkeley.edu

- Integral Lipschitz graph filters are $\Rightarrow$ (i) Arbitrarily sharp at eigenvalues $\lambda \to 0$
  $\Rightarrow$ (ii) Flat at eigenvalues $\lambda \to \infty$
- They are uniformly Lipschitz stable $\Rightarrow$ (i) For $\lambda \to 0$ eigenvalue change bounded
  $\Rightarrow$ (ii) For $\lambda \to \infty$ filter change bounded.

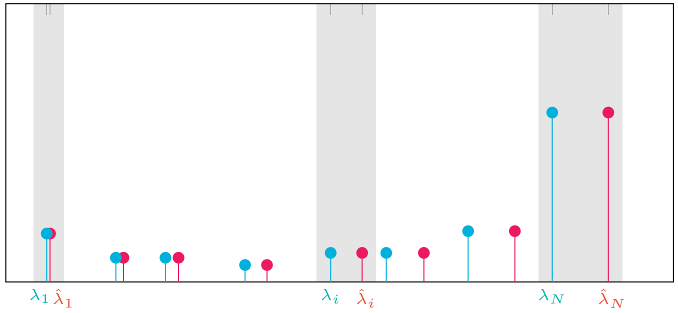


Should this work? Stability to Perturbations of the Graph

F. Gama
fgama@berkeley.edu

- Integral Lipschitz filters are stable, but they can't discriminate high frequency graph signals
- Pointwise nonlinearities are low-pass demodulators that generate low frequency components

$\Rightarrow$ That can separated by stable filters at the next layer



Quiz

F. Gama
fgama@berkeley.edu

What? $\Rightarrow$ Machine learning on graphs

Why? $\Rightarrow$ Generic models of signal structure. Models of physical infrastructure

How? $\Rightarrow$ Graph Signals. Graph Convolutions. Graph Neural Networks.

Does this work? $\Rightarrow$ Recommendation systems. Distributed collaborative intelligent systems

Large scale physical infrastructure

Should this work? $\Rightarrow$ Permutation equivariance. Stability to graph perturbations.

Which graph filters can't be if we push their discriminative power.