Network Science (VU) (706.703)

Graph Filters and Graph Neural Networks

Denis Helic

HCC, TU Graz

December 2, 2025

Outline

Slides

Slides are partially based on paper Graph Filters for Signal Processing and Machine Learning on Graphs.

Introduction

Filters

- Filters are information processing architectures
- Goal: preserve only relevant information
- In ML, filters extract relevant patterns
- In addition, filters serve as an inductive bias
- Examples:
    - PCA is a low-pass filter for the correlation matrix
    - Convolution filters (CNNs) exploit structural invariance

Graph Filters

- Graph filters are processing units for graph signals
- Typically, graph filters are linear operators
- We can think of graph filters as parametric functions of the input

$$\mathbf{y} = \mathbf{f}(\mathbf{x}; \boldsymbol{\theta})$$

- $\mathbf{x}$: input, $\boldsymbol{\theta}$: parameters, $\mathbf{y}$: output, $\mathbf{f}$: linear function
- Graph filters have a spectral interpretation
- Equivariant to permutations (relabeling)

Graph Filters: Applications and Tasks

- Signal reconstruction, i.e., denoising
- Signal compression, i.e., low-rank embeddings
- Signal classification/regression
- Node classification/regression
- Graph classification/regression
- Link prediction
- Graph learning

Graph Filters

Signal Processing on Graphs

Graph Shift Operator

- Graph shift operator (GSO) is a generic operator $\mathbf{S}$
- Typically, $\mathbf{S}$ is a symmetric weighted matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$
- GSO is the basic operator for constructing graph filters
- Many possibilities for GSO (depending on the task):
  - $\mathbf{A}$, $\mathbf{L}$
  - $\mathbf{A}_n = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, $\mathbf{L}_n = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$
  - $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}(\mathbf{I} + \mathbf{A})\mathbf{D}^{-1/2}$
  - $\mathbf{L}_{rw} = \mathbf{D}^{-1}\mathbf{L}$ (asymmetric)
  - $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ (asymmetric)

Graph Signal

- Graph signal is a function from set of nodes onto set of real numbers
- Every node is associated with a real value x_i
- We collect all the values into a vector $\mathbf{x} \in \mathbb{R}^n$
- For example, the values may represent
  - Number of postings from users in a social network
  - Ratings of users for a specific item
- Jupyter Notebook example: signals.ipynb

Graph Convolutional Filters

- Convolutional filters are basic building blocks in ML
- Convolutional neural networks
- Computationally efficient as they share parameters
- Leverage symmetries in the domain

Graph Convolutional Filter: Basic Block

- Convolutional filter performs shift-and-sum operation
- Signal shift:

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

- Case $\mathbf{S} = \mathbf{A}$: one step propagation and sum over neighbors
- Case $\mathbf{S} = \mathbf{L}$: one step propagation and sum of differences between a given node and its neighbors
  - How the value at current node differs from the average value of neighbors

Graph Convolutional Filter

- Given a set of parameters $\mathbf{h} \in \mathbb{R}^{K+1}$
- Graph convolutional filter is a linear combination of K shifted signals:

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

- $\mathbf{H}(\mathbf{S}) \in \mathbb{R}^{n \times n}$ is the polynomial filtering matrix

Graph Convolutional Filters: Properties

- Linearity:

$$\alpha \mathbf{H}(\mathbf{S})\mathbf{x}_1 + \beta \mathbf{H}(\mathbf{S})\mathbf{x}_2 = \mathbf{H}(\mathbf{S})(\alpha \mathbf{x}_1 + \beta \mathbf{x}_2)$$

- Shift invariance:

$$\mathbf{S}\mathbf{H}(\mathbf{S}) = \mathbf{H}(\mathbf{S})\mathbf{S} \implies \mathbf{H}_1(\mathbf{S})\mathbf{H}_2(\mathbf{S})\mathbf{x} = \mathbf{H}_2(\mathbf{S})\mathbf{H}_1(\mathbf{S})\mathbf{x}$$

Graph Convolutional Filters: Properties

- Permutation equivariance:
  - Permutation (relabeling) matrices:

$$\mathbf{P} \in \{0,1\}^{n\times n} : \mathbf{P1} = \mathbf{1}, \mathbf{P}^T\mathbf{1} = \mathbf{1}$$

  - Relabeling: $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{S}\mathbf{P}$ and $\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{x}$
  - Relabeling and filtering is equivalent to filtering and then relabeling:

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

Graph Convolutional Filters: Properties

- Parameter sharing allow for inductive processing:
    - Learn filters on one graph and apply on another
- Locality:
    - Shifted signals propagate locally in the neighborhood
    - But we can proceed to further hops and include global information
    - Supports computational efficiency
    - Supports parameters sharing between neighborhoods

Spectral Analysis

Graph Frequency Response

Graph Fourier Transform

- Given $\mathbf{S}$ symmetric we have eigenvector decomposition $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$
- $\mathbf{U}$: orthonormal eigenvector matrix with eigenvectors in columns
- $\mathbf{\Lambda}$: diagonal matrix of eigenvalues
- Graph Fourier Transform (GFT) of signal $\mathbf{x}$ is given by:

$$\tilde{\mathbf{x}} = \mathbf{U}^T\mathbf{x}$$

- Inverse GFT of $\tilde{\mathbf{x}}$ is given by:

$$\mathbf{x} = \mathbf{U}\tilde{\mathbf{x}}$$

Graph Frequency Response

- Given a graph convolutional filter:

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

- GFT of input: $\tilde{\mathbf{x}} = \mathbf{U}^T \mathbf{x}$
- GFT of output: $\tilde{\mathbf{y}} = \mathbf{U}^T \mathbf{y}$
- We have:

$$\tilde{\mathbf{y}} = \sum_{k=0}^{K} h_k \mathbf{\Lambda}^k \tilde{\mathbf{x}}$$

- Proof by substituting $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ in the convolutional filter and expanding

Graph Frequency Response

- Individual components:

$$\tilde{y}_i = \sum_{k=0}^{K} h_k \lambda_i^k \tilde{x}_i = \tilde{h}(\lambda_i)\tilde{x}_i$$

- Graph convolutions are pointwise in the GFT domain
- Frequency response of a graph filter:

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

- Graph filters are diagonal matrices in the frequency domain:

$$\tilde{\mathbf{H}} = diag(\tilde{h}(\lambda))$$

Graph Frequency Response

- Graph frequency response $\tilde{h}(\lambda)$ is a polynomial in λ
- It is independent of the graph
- Graph only induces a certain response through its eigenvalues

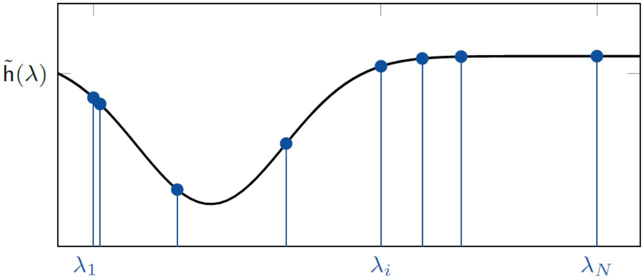

Figure: From Graph Filters for Signal Processing and Machine Learning on Graphs

Applications in Filter Design

- Sometimes we want a specific response
- For example, when removing noise (denoising)
- We may want to use low-pass filter
- We then simply set:

$$\tilde{\mathbf{H}} = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & 0 \end{pmatrix}$$

- ... and apply GFT
- Jupyter Notebook example: gft.ipynb & propagation.ipynb

Filter Learning

Learning Filter Parameters

Learning Filters from Data

- Often, we do not know exact filters and have only input-output pairs
- We will learn filters from data using machine learning (ML)
- We start with an example application
- We have a noisy signal and would like to reconstruct the original signal
- Signal denoising

Signal Denoising

- We observe a noisy signal:

$$\mathbf{y} = \mathbf{x} + \boldsymbol{\epsilon}$$

- $\boldsymbol{\epsilon}$ is noise
- The goal is to reconstruct the true signal $\mathbf{x}$ from the noisy signal $\mathbf{y}$

Signal Denoising: Key Assumptions

- Smoothness: signal $\mathbf{x}$ is smooth
  - Like connect to like
  - Homophily
- Noise is uncorrelated with the graph structure

Signal Denoising: Approaches

- Use GFT with a low-pass filter
- Optimize a global function:

$$L(\mathbf{x}) = ||\mathbf{x} - \mathbf{y}||_2^2 + \alpha \mathbf{x}^T \mathbf{L} \mathbf{x}$$

- $||\mathbf{x} - \mathbf{y}||_2^2$ is data fidelity term
- $\mathbf{x}^T \mathbf{L} \mathbf{x}$ is the smoothness term
  - This is Laplacian smoothness regularization
- α controls the trade-off between fidelity and regularization

Signal Denoising: Laplacian Smoothness

$$
\begin{aligned}
L(\mathbf{x}) &= ||\mathbf{x}-\mathbf{y}||_2^2 + \alpha\mathbf{x}^T\mathbf{L}\mathbf{x} \\
&= (\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y}) + \alpha\mathbf{x}^T\mathbf{L}\mathbf{x} \\
\frac{\partial L}{\partial \mathbf{x}} &= 2(\mathbf{x}-\mathbf{y}) + 2\alpha\mathbf{L}\mathbf{x} = 0 \\
& \dots \\
\mathbf{x} &= (\mathbf{I}+\alpha\mathbf{L})^{-1}\mathbf{y}
\end{aligned}
$$

- Low-pass filter: $\tilde{h}(\lambda) = \frac{1}{1+\alpha\lambda}$

Signal Denoising: Total Variation

- Alternatively, we could use total variation as the regularization term:

$$L(\mathbf{x}) = ||\mathbf{x} - \mathbf{y}||_2^2 + \alpha \sum_{i \sim j} |x_i - x_j|$$

- We can then optimize numerically
- Jupyter notebook example: denoising.ipynb

Graph Neural Networks

Machine Learning on Graphs

Graph Neural Networks

- Graph neural networks (GNNs) are non-linear layered filter architectures
- Non-linearity allows GNNs to capture more complex relationships
- Compositional form allows for sequential extraction of features
- This improves capabilities over linear and non-linear filters

GNN

- The basic building block of GNNs is graph perceptron
- It is a non-linear mapping of a nested graph filter

$$\mathbf{y} = \sigma(\mathbf{H}(\mathbf{x}))$$

- $\mathbf{H}(\mathbf{x})$ is a linear graph filter
  - For example: $\mathbf{H}(\mathbf{x}) = h_1 \mathbf{S}\mathbf{x}$
- σ is a element-wise non-linear activation function
  - ReLU, tanh, sigmoid, ...

GNN

- Cascading perceptrons gives rise to a GNN

$$\boldsymbol{\Phi}(\mathbf{x}) = \mathbf{x}_\ell$$

- With $\mathbf{x}_\ell = \sigma(\mathbf{H}_\ell(\mathbf{x}_{\ell-1}))$
- Input to the GNN is the graph signal which is processed to get $\mathbf{x}_1$
- Then $\mathbf{x}_1$ is the input to the next layer to obtain $\mathbf{x}_2$
- This continues until the last layer where we get $\mathbf{x}_\ell$

GNN

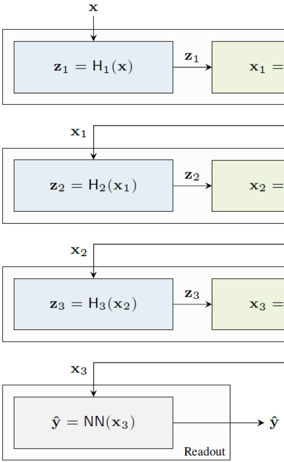

Figure: From Graph Filters for Signal Processing and Machine Learning on Graphs

GNN vs. Filter Properties

- Most of GNNs keep some of the filter properties
- Permutation equivariance
- Parameter sharing, i.e., inductive learning
- Locality

Multidimensional Features

- Representational power of GNNs is increased with multidimensional features
- We use feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$
- d is the number of features
- We then also need multidimensional parameters, i.e, parameter matrices $\mathbf{W}$
- Typically, at each layer: $\mathbf{W}_\ell \in \mathbb{R}^{d_1 \times d_2}$
- d_1 at the first layer equals d
- d_2 at the last layer defines the number of output features

Readout Layer

- If the output dimension do not match the target output $\mathbf{y}$ we use a readout layer
- Decode the GNN embeddings into the final output
- For example, binary classification has only two labels
- Typically, the readout layer also has learnable parameters
- Jupyter notebook example: cluster.ipynb

Graph Convolutional Neural Networks

- Most popular GNN architecture: GCNs

$$\mathbf{X}_\ell = \sigma(\mathbf{S}\mathbf{X}_{\ell-1}\mathbf{W}_\ell)$$

- Comparing to filters:

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

- This is just one hop propagation
- We can extend it to multi-hop propagation:

$$\mathbf{X}_\ell = \sigma(\sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{X}_{\ell-1}\mathbf{W}_{\ell k})$$