Spectral Graph Convolutional Networks

ACMS 80770: Deep Learning with Graphs

Instructor: Navid Shervani-Tabar

Department of Applied and Comp Math and Stats

UNIVERSITY OF NOTRE DAME

Fourier transform

- In the previous lecture, we explored the relation between the **Fourier transform** and the **convolution** operation.

Fourier transform

- In the previous lecture, we explored the relation between the **Fourier transform** and the **convolution** operation.
- We discussed that **Fourier transform** takes a data residing on the Euclidean space and maps it to the Fourier domain

$$\mathcal{F}(f(t)) = \hat{f}(s) := \langle f, e^{2\pi ist} \rangle = \int_{\mathbb{R}} f(t) e^{-2\pi ist} dt$$

where $e^{2\pi ist}$ represents the **Fourier basis** and $\hat{f}(s)$ is the corresponding **Fourier coefficient**.

Fourier transform

- In the previous lecture, we explored the relation between the **Fourier transform** and the **convolution** operation.
- We discussed that **Fourier transform** takes a data residing on the Euclidean space and maps it to the Fourier domain

$$\mathcal{F}(f(t)) = \hat{f}(s) := \langle f, e^{2\pi ist} \rangle = \int_{\mathbb{R}} f(t) e^{-2\pi ist} dt$$

where $e^{2\pi ist}$ represents the **Fourier basis** and $\hat{f}(s)$ is the corresponding **Fourier coefficient**.

- Given $\hat{f}(s)$, one can recover the function f by projecting $\hat{f}(s)$ back to the Euclidean domain using **inverse Fourier transform**.

$$\mathcal{F}^{-1}(\hat{f}(s)) = f(t) := \int_{\mathbb{R}} \hat{f}(s) e^{2\pi ist} ds$$

Fourier Transform

- We recall that the **Laplace operator** is defined as

$$\Delta f(t) = \nabla^2 f(t) = \frac{\partial^2 f}{\partial t^2}$$

- We can show that **bases** of the Fourier domain are **eigenfunctions** of the Laplace operator

$$-\Delta \left(e^{2\pi i s t}\right) = -\frac{\partial^2}{\partial t^2} \left(e^{2\pi i s t}\right) = (2\pi s)^2 e^{2\pi i s t}$$

Graph Fourier Transform

- We recall that the **Laplace operator** is defined as

$$\Delta f(t) = \nabla^2 f(t) = \frac{\partial^2 f}{\partial t^2}$$

- We can show that **bases** of the Fourier domain are **eigenfunctions** of the Laplace operator

$$-\Delta \left(e^{2\pi i s t}\right) = -\frac{\partial^2}{\partial t^2} \left(e^{2\pi i s t}\right) = (2\pi s)^2 e^{2\pi i s t}$$

- We can exploit the analogy between the **Laplacian** matrix and the **Laplace** operator to extend the Fourier transform to graphs.

Graph Fourier Transform

- We recall that the **Laplace operator** is defined as

$$\Delta f(t) = \nabla^2 f(t) = \frac{\partial^2 f}{\partial t^2}$$

- We can show that **bases** of the Fourier domain are **eigenfunctions** of the Laplace operator

$$-\Delta \left(e^{2\pi i s t}\right) = -\frac{\partial^2}{\partial t^2}\left(e^{2\pi i s t}\right) = (2\pi s)^2 e^{2\pi i s t}$$

- We can exploit the analogy between the **Laplacian** matrix and the **Laplace** operator to extend the Fourier transform to graphs.

- To that end, one can define the **graph Fourier bases** as the **eigenvectors** $\{u_\ell\}_{\ell=0,\ldots,|V|-1}$ of the Laplacian matrix

$$\mathbf{L} u_\ell = \lambda_\ell u_\ell$$

Graph Fourier Transform

- Using the eigenvectors U as Fourier bases, we can define the **Fourier transform** of a signal $\mathbf{f} \in \mathbb{R}^N$ in the **graph domain** as

$$\hat{f}\left(\lambda_\ell\right) := \langle f, u_\ell \rangle = \sum_{i=1}^{N} f(i) u_\ell(i)$$

- In the matrix form

$$\hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f}$$

Graph Fourier Transform

- Using the eigenvectors U as Fourier bases, we can define the **Fourier transform** of a signal $\mathbf{f} \in \mathbb{R}^N$ in the **graph domain** as

$$\hat{f}\,(\lambda_\ell) := \langle f, u_\ell \rangle = \sum_{i=1}^{N} f(i) u_\ell(i)$$

- In the matrix form

$$\hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f}$$

- Analogously, one can define the **inverse graph Fourier transform** as

$$f(i) = \sum_{\ell=0}^{N-1} \hat{f}\,(\lambda_\ell)\, u_\ell(i)$$

- In the matrix notation

$$\mathbf{f} = \mathbf{U}\hat{\mathbf{f}}$$

Spectral Graph Convolution

- Like the Euclidean domain, we can use the **graph Fourier** transform to represent **convolution** on graphs in the spectral domain.

Spectral Graph Convolution

- Like the Euclidean domain, we can use the **graph Fourier** transform to represent **convolution** on graphs in the spectral domain.
- This is shown by the Hadamard product

$$\mathbf{f} *_G \mathbf{h} = \mathbf{U}\left(\mathbf{U}^\top \mathbf{f} \odot \mathbf{U}^\top \mathbf{h}\right)$$

where $*_G$ denotes a convolution operator specific to the graph G, and $\mathbf{U}$ is eigenvector of $\mathbf{L}$.

Spectral Graph Convolution

- Like the Euclidean domain, we can use the **graph Fourier** transform to represent **convolution** on graphs in the spectral domain.

- This is shown by the Hadamard product

$$\mathbf{f} *_G \mathbf{h} = \mathbf{U}\left(\mathbf{U}^\top \mathbf{f} \odot \mathbf{U}^\top \mathbf{h}\right)$$

  where $*_G$ denotes a convolution operator specific to the graph G, and $\mathbf{U}$ is eigenvector of $\mathbf{L}$.

- Note that since the Fourier transform on graphs is defined using the **eigenvectors** of Laplacian $\mathbf{L}$ of the graph, the transform is **specific to the graph** G.

- Therefore, convolution operator $*_G$ is defined for the graph G.

Spectral Graph Convolution

- In the **spectral representation** of the convolution,

$$\mathbf{f} *_G \mathbf{h} = \mathbf{U}\left(\mathbf{U}^\top \mathbf{f} \odot \mathbf{U}^\top \mathbf{h}\right)$$

The term $\mathbf{U}^{\mathrm{T}}\mathbf{h}$ transforms the filter $\mathbf{h}$, which is defined in the spatial domain, to the spectral domain.

Spectral Graph Convolution

- In the **spectral representation** of the convolution,

$$\mathbf{f} *_G \mathbf{h} = \mathbf{U}\left(\mathbf{U}^\top \mathbf{f} \odot \mathbf{U}^\top \mathbf{h}\right)$$

The term $\mathbf{U}^{\mathrm{T}}\mathbf{h}$ transforms the filter $\mathbf{h}$, which is defined in the spatial domain, to the spectral domain.

- In index notation, this is represented as

$$(f * h)(i) := \sum_{\ell=0}^{N-1} \hat{f}(\lambda_\ell)\hat{h}(\lambda_\ell)u_\ell(i)$$

where $\{u_\ell\}_{\ell=0,\ldots,N-1}$ is the set of eigenvectors of the Laplacian matrix and $\hat{f}$ and $\hat{h}$ are spectral representations of the signal f and filter h, respectively.

Spectral Filters

- Alternatively, one can **directly** define the filter in the **spectral domain** of the graph.

Spectral Filters

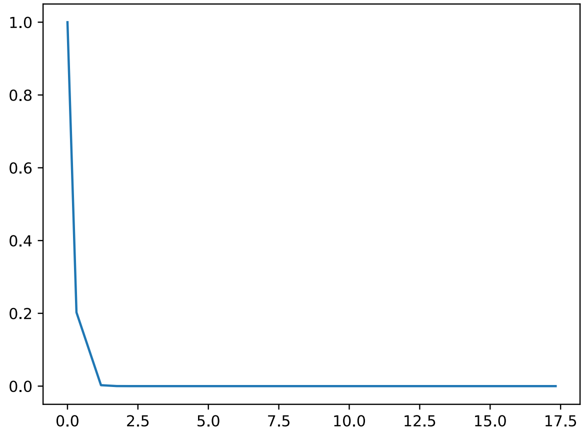
- Alternatively, one can **directly** define the filter in the **spectral domain** of the graph.
  - Heat kernel is defined as

$$\hat{h}\left(\lambda_\ell\right) = \exp\left(-5\lambda_\ell\right)$$

Spectral Filters

- Alternatively, one can **directly** define the filter in the **spectral domain** of the graph.
  - Heat kernel is defined as

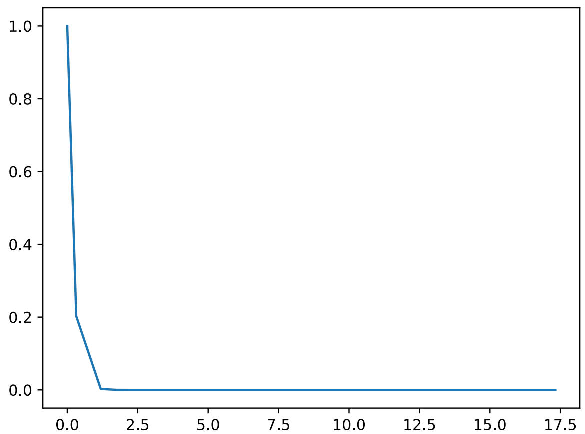
$$\hat{h}\left(\lambda_\ell\right) = \exp\left(-5\lambda_\ell\right)$$

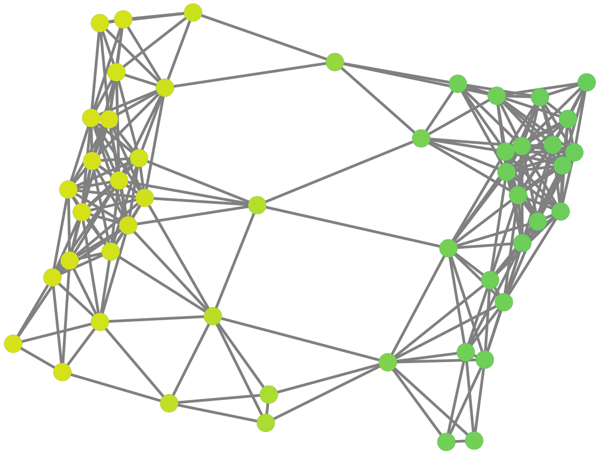


Spectral Filters

- Alternatively, one can **directly** define the filter in the **spectral domain** of the graph.
  - Heat kernel is defined as

$$\hat{h}\left(\lambda_\ell\right) = \exp\left(-5\lambda_\ell\right)$$





Spectral Filters

- Alternatively, one can **directly** define the filter in the **spectral domain** of the graph.
  - Heat kernel is defined as

$$\hat{h}\left(\lambda_\ell\right) = \exp\left(-5\lambda_\ell\right)$$

- This approach enables us to define the **convolution filter** directly in the **spectral domain**.

$$\mathbf{f} *_G \mathbf{h} = \mathbf{U}\left(\mathbf{U}^\top \mathbf{f} \odot \theta_h\right)$$

where

$$\theta_h = \mathbf{U}^T \mathbf{h} \in \mathbb{R}^{|V|}$$

Spectrum-based Methods

- This spectral representation of the convolution can be used to define **trainable convolution layers** on graph.

Spectrum-based Methods

- This spectral representation of the convolution can be used to define **trainable convolution layers** on graph.
- Let θ_h be a **non-parametric filter**; that is all parameters in the filter are free

$$\begin{aligned} \mathbf{f} *_G \mathbf{h} &= \mathbf{U}\left(\mathbf{U}^\top \mathbf{f} \odot \theta_h\right) \\ &= \mathbf{U}\left(\mathrm{diag}\left(\theta_h\right) \mathbf{U}^\top \mathbf{f}\right) \end{aligned}$$

where $\mathrm{diag}(\theta_h) \in \mathbb{R}^{|V| \times |V|}$ is a diagonal matrix of the graph Fourier coefficients of the filter.

Spectrum-based Methods

- This spectral representation of the convolution can be used to define **trainable convolution layers** on graph.
- Let θ_h be a **non-parametric filter**; that is all parameters in the filter are free

$$\begin{aligned} \mathbf{f} *_G \mathbf{h} &= \mathbf{U}\left(\mathbf{U}^\top \mathbf{f} \odot \theta_h\right) \\ &= \mathbf{U}\left(\mathrm{diag}\left(\theta_h\right) \mathbf{U}^\top \mathbf{f}\right) \end{aligned}$$

where $\mathrm{diag}(\theta_h) \in \mathbb{R}^{|V|\times|V|}$ is a diagonal matrix of the graph Fourier coefficients of the filter.

- Due to their dependence on the domain of the graph (through the eigenvectors) models using such convolutional layers are referred to as **spectrum-based methods**.

Spectral Convolutional Neural Networks

- **Spectral Convolutional Neural Network** (SCNN) layers define convolution layers as

$$\boldsymbol{H}_{:,j}^{(t+1)} = \sigma\left(\sum_{i=1}^{d_\ell} \mathbf{U}_K \mathrm{diag}(\theta^{(t)})_{i,j} \mathbf{U}_K^\top \boldsymbol{H}_{:,i}^{(t)}\right)$$

with $1 \leq j \leq d_{t+1}$ and $1 \leq i \leq d_t$, σ is non-linearity, and $diag(\theta)_{i,j} \in \mathbb{R}^{K \times K}$ are trainable diagonal spectral filters.

Spectral Convolutional Neural Networks

- **Spectral Convolutional Neural Network** (SCNN) layers define convolution layers as

$$\boldsymbol{H}^{(t+1)}_{:,j} = \sigma\left(\sum_{i=1}^{d_\ell} \mathbf{U}_K \mathrm{diag}(\theta^{(t)})_{i,j} \mathbf{U}_K^\top \boldsymbol{H}^{(t)}_{:,i}\right)$$

  with $1 \leq j \leq d_{t+1}$ and $1 \leq i \leq d_t$, σ is non-linearity, and $diag(\theta)_{i,j} \in \mathbb{R}^{K\times K}$ are trainable diagonal spectral filters.

- Note that, only top K eigenvectors $\boldsymbol{U}_K \in \mathbb{R}^{|V|\times K}$ of the Laplacian L are used as they carry the **most informative** data.

Spectral Convolutional Neural Networks

- **Spectral Convolutional Neural Network** (SCNN) layers define convolution layers as

$$\boldsymbol{H}_{:,j}^{(t+1)} = \sigma\left(\sum_{i=1}^{d_\ell} \mathbf{U}_K \mathrm{diag}(\theta^{(t)})_{i,j} \mathbf{U}_K^\top \boldsymbol{H}_{:,i}^{(t)}\right)$$

  with $1 \leq j \leq d_{t+1}$ and $1 \leq i \leq d_t$, σ is non-linearity, and $diag(\theta)_{i,j} \in \mathbb{R}^{K \times K}$ are trainable diagonal spectral filters.

- Note that, only top K eigenvectors $\boldsymbol{U}_K \in \mathbb{R}^{|V| \times K}$ of the Laplacian L are used as they carry the **most informative** data.

- Due to their spectrum-based nature, these methods can only be used in the **transductive** setting.

Spectrum-free Methods

- One problem with such a definition is that $\mathrm{diag}(\theta_h)$ has no dependency on the **structure** if the graph.
- This may result in filters that are **arbitrarily non-local** with respect to the nodes.

Spectrum-free Methods

- One problem with such a definition is that $\text{diag}(\theta_h)$ has no dependency on the **structure** if the graph.
- This may result in filters that are **arbitrarily non-local** with respect to the nodes.
- To remedy that, instead, one can use a **polynomial parametrization** based on the spectrum of the graph.
- To that end, we can approximate the spectral filter as a **polynomial expansion** of the graph spectrum

Spectrum-free Methods

- One problem with such a definition is that $\mathrm{diag}(\theta_h)$ has no dependency on the **structure** if the graph.
- This may result in filters that are **arbitrarily non-local** with respect to the nodes.
- To remedy that, instead, one can use a **polynomial parametrization** based on the spectrum of the graph.
- To that end, we can approximate the spectral filter as a **polynomial expansion** of the graph spectrum

$$p_K(\mathbf{\Lambda}) = \sum_{k=0}^{K} \theta_k \mathbf{\Lambda}^k$$

which represents a polynomial of degree K with respect to the eigenvalues of the Laplacian $\mathbf{L}$.

Spectrum-free Methods

- Thus, we can **reformulate** the convolution as

$$\mathbf{f} *_G \mathbf{h} = \left(\mathbf{U} p_K(\mathbf{\Lambda}) \mathbf{U}^\top \right) \mathbf{f}$$

Spectrum-free Methods

- Thus, we can **reformulate** the convolution as

$$\mathbf{f} *_G \mathbf{h} = \left(\mathbf{U} p_K(\mathbf{\Lambda}) \mathbf{U}^\top \right) \mathbf{f}$$

- By interpreting the eigenvalues as analogs to the **frequency**, we can interpret $p_K(\mathbf{\Lambda})$ as the **filter frequency response**.

Spectrum-free Methods

- Thus, we can **reformulate** the convolution as

$$\mathbf{f} *_G \mathbf{h} = \left(\mathbf{U} p_K(\mathbf{\Lambda}) \mathbf{U}^\top \right) \mathbf{f}$$

- By interpreting the eigenvalues as analogs to the **frequency**, we can interpret $p_K(\mathbf{\Lambda})$ as the **filter frequency response**.
- One drawback with this representation of the convolution is that it requires us to perform **eigendecomposition** of the Laplacian matrix.
- For large graphs, such an operation may be prohibitively **expensive**.
  - Social networks.

Spectrum-free Methods

- Noting that

$$\left(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top\right)^k = \mathbf{U}\left(\mathbf{\Lambda}\right)^k\mathbf{U}^\top$$

we can show that this polynomial parameterization may be reformulated as a **polynomial function** of the **Laplacian** matrix

$$\mathbf{U}p_K(\mathbf{\Lambda})\mathbf{U}^\top = p_K\left(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top\right) = p_K(\mathbf{L})$$

Spectrum-free Methods

- Noting that

$$\left(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\top}\right)^{k} = \mathbf{U}\left(\mathbf{\Lambda}\right)^{k}\mathbf{U}^{\top}$$

  we can show that this polynomial parameterization may be reformulated as a **polynomial function** of the **Laplacian** matrix

$$\mathbf{U}p_K(\mathbf{\Lambda})\mathbf{U}^{\top} = p_K\left(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\top}\right) = p_K(\mathbf{L})$$

- We can see that defining a filtering matrix as a degree k polynomial of the Laplacian constructs a $\boldsymbol{k}$ $-$**localized** filtering.
- Therefore, parametrizing filter with eigenvalues $\mathbf{\Lambda}$ results in **localized filters**.

Spectral Graph-based Neural Networks

- We can exploit the notation of **spectral convolution** on graphs to propose convolutional graph neural **networks**, based on the **parameterization** of the **filters**.

Spectral Graph-based Neural Networks

- We can exploit the notation of **spectral convolution** on graphs to propose convolutional graph neural **networks**, based on the **parameterization** of the **filters**.
- These networks use either the **non-parametric spectral filters**

$$\mathbf{f} *_G \mathbf{h} = \left(\mathbf{U} \, \mathrm{diag}\left(\theta_h\right) \mathbf{U}^\top \right) \mathbf{f}$$

Spectral Graph-based Neural Networks

- We can exploit the notation of **spectral convolution** on graphs to propose convolutional graph neural **networks**, based on the **parameterization** of the **filters**.
- These networks use either the **non-parametric spectral filters**

$$\mathbf{f} *_G \mathbf{h} = \left(\mathbf{U} \, \mathrm{diag}\left(\theta_h\right) \mathbf{U}^\top \right) \mathbf{f}$$

Or **parametric spectral filters**

$$\mathbf{f} *_G \mathbf{h} = \left(\mathbf{U} p_K(\mathbf{\Lambda}) \mathbf{U}^\top \right) \mathbf{f}$$

Spectral Graph-based Neural Networks

- We can exploit the notation of **spectral convolution** on graphs to propose convolutional graph neural **networks**, based on the **parameterization** of the **filters**.
- These networks use either the **non-parametric spectral filters**

$$\mathbf{f} *_G \mathbf{h} = \left(\mathbf{U} \operatorname{diag}(\theta_h) \mathbf{U}^\top \right) \mathbf{f}$$

  Or **parametric spectral filters**

$$\mathbf{f} *_G \mathbf{h} = \left(\mathbf{U} p_K(\mathbf{\Lambda}) \mathbf{U}^\top \right) \mathbf{f}$$

- Combined with **non-linear** layers and **stack** them to build deep graph-based neural networks.

ChebNet

- One can use different **polynomial basis** to define the parametric filters.
- One such model, **ChebNet**, uses a **Chebyshev polynomial** to approximate the $p_K(\boldsymbol{L})$.

ChebNet

- One can use different **polynomial basis** to define the parametric filters.
- One such model, **ChebNet**, uses a **Chebyshev polynomial** to approximate the $p_K(\boldsymbol{L})$.
- A **Chebyshev polynomial** of order K is computed through the recursive relation

$$T_k(\lambda) = 2\lambda T_{k-1}(\lambda) - T_{k-2}(\lambda)$$

where

$$T_0(\lambda) = 1,$$
$$T_1(\lambda) = \lambda.$$

ChebNet

- Chebyshev polynomials define **orthonormal basis** in the interval $[-1,1]$.
- We can **parametrize** the filter $p_K(\mathbf{\Lambda})$ using the Chebyshev polynomials as

$$p_\theta(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{\Lambda}})$$

where $\boldsymbol{\theta} \in \mathbb{R}^K$ is the vector of **polynomial coefficients**, and $T_K(\widetilde{\mathbf{\Lambda}})$ is a Chebyshev **polynomial** of order K.

ChebNet

- Chebyshev polynomials define **orthonormal basis** in the interval $[-1,1]$.
- We can **parametrize** the filter $p_K(\mathbf{\Lambda})$ using the Chebyshev polynomials as

$$p_\theta(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{\Lambda}})$$

  where $\boldsymbol{\theta} \in \mathbb{R}^K$ is the vector of **polynomial coefficients**, and $T_K(\widetilde{\mathbf{\Lambda}})$ is a Chebyshev **polynomial** of order K.
- Note that, in order for polynomials to form orthonormal basis, the **eigenvalues** are **normalized** as

$$\tilde{\mathbf{\Lambda}} = \frac{2\mathbf{\Lambda}}{\lambda_{\max}} - I_{|V|}$$

  to map them from the interval $[0, \lambda_{\max}]$ to $[-1,1]$.

ChebNet

- Using this expansion, one can represent the **convolution** as

$$\hat{\mathbf{f}} = p_K(\mathbf{L})\mathbf{f} = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{f}$$

where $\tilde{\mathbf{L}}$ is the **scaled normalized Laplacian** matrix

$$\tilde{\boldsymbol{L}} = \frac{2\boldsymbol{L}}{\lambda_{\max}} - I_{|V|}$$

ChebNet

- Using this expansion, one can represent the **convolution** as

$$\hat{\mathbf{f}} = p_K(\mathbf{L})\mathbf{f} = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{f}$$

where $\tilde{\mathbf{L}}$ is the **scaled normalized Laplacian** matrix

$$\tilde{\boldsymbol{L}} = \frac{2\boldsymbol{L}}{\lambda_{\max}} - I_{|V|}$$

- Thus, each layer of ChebNet implements

$$\boldsymbol{H}^{(t+1)} = \sigma\left(\sum_{k=0}^{K} T_k(\tilde{\mathbf{L}})\boldsymbol{H}^{(t)}\Theta_k^{(t)}\right)$$

where $\boldsymbol{H}^{(t)} \in \mathbb{R}^{|V|\times d}$ and $\Theta_k \in \mathbb{R}^{d\times d'}$.

Graph Convolutional Network

- One can **simplify** the ChebNet model by limiting the polynomial order to $K = 1$.
- This yields a **first order approximation** of the convolution operation in ChebNet.

Graph Convolutional Network

- One can **simplify** the ChebNet model by limiting the polynomial order to $K = 1$.
- This yields a **first order approximation** of the convolution operation in ChebNet.
- We can derive this as

$$\mathbf{f} *_G \mathbf{h} = p_K(\tilde{L})\mathbf{f}$$

Graph Convolutional Network

- One can **simplify** the ChebNet model by limiting the polynomial order to $K = 1$.
- This yields a **first order approximation** of the convolution operation in ChebNet.
- We can derive this as

$$\mathbf{f} *_G \mathbf{h} = p_K(\tilde{L})\mathbf{f}$$

$$= \sum_{k=0}^{1} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{f}$$

Graph Convolutional Network

- One can **simplify** the ChebNet model by limiting the polynomial order to $K = 1$.
- This yields a **first order approximation** of the convolution operation in ChebNet.
- We can derive this as

$$\begin{aligned}\mathbf{f} *_G \mathbf{h} &= p_K(\tilde{L})\mathbf{f} \\ &= \sum_{k=0}^{1} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{f} \\ &= \theta_0 \mathbf{f} + \theta_1 \tilde{\mathbf{L}}\mathbf{f}\end{aligned}$$

Graph Convolutional Network

- One can **simplify** the ChebNet model by limiting the polynomial order to $K = 1$.
- This yields a **first order approximation** of the convolution operation in ChebNet.
- We can derive this as

$$\begin{aligned}\mathbf{f} *_G \mathbf{h} &= p_K(\tilde{L})\mathbf{f} \\ &= \sum_{k=0}^{1} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{f} \\ &= \theta_0 \mathbf{f} + \theta_1 \tilde{\mathbf{L}}\mathbf{f} \\ &= \theta_0 \mathbf{f} + \theta_1 \left[\frac{2\mathbf{L}}{\lambda_{\max}} - I_{|V|} \right] \mathbf{f}\end{aligned}$$

Graph Convolutional Network

- Since eigenvalues of $\boldsymbol{L}$ fall in the interval $[0,2]$, we can approximate $\lambda_{\max} = 2$ and write

$$\mathbf{f} *_G \mathbf{h} = \theta_0 f + \theta_1 \left[\mathbf{L} - I_{|V|}\right] \mathbf{f}$$

Graph Convolutional Network

- Since eigenvalues of $\boldsymbol{L}$ fall in the interval $[0,2]$, we can approximate $\lambda_{\max} = 2$ and write

$$
\begin{aligned}
\mathbf{f} *_G \mathbf{h} &= \theta_0 f + \theta_1 \left[\mathbf{L} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \left(\mathbf{D} - \mathbf{A}\right) \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f}
\end{aligned}
$$

Graph Convolutional Network

- Since eigenvalues of $\boldsymbol{L}$ fall in the interval $[0,2]$, we can approximate $\lambda_{\max} = 2$ and write

$$
\begin{aligned}
\mathbf{f} *_G \mathbf{h} &= \theta_0 f + \theta_1 \left[\mathbf{L} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \left(\mathbf{D} - \mathbf{A}\right) \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \mathbf{D} \mathbf{D}^{-\frac{1}{2}} - \mathbf{D}^{-\frac{1}{2}} A \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f}
\end{aligned}
$$

Graph Convolutional Network

- Since eigenvalues of $\boldsymbol{L}$ fall in the interval $[0,2]$, we can approximate $\lambda_{\max} = 2$ and write

$$
\begin{aligned}
\mathbf{f} *_G \mathbf{h} &= \theta_0 f + \theta_1 \left[\mathbf{L} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \left(\mathbf{D} - \mathbf{A}\right) \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \mathbf{D} \mathbf{D}^{-\frac{1}{2}} - \mathbf{D}^{-\frac{1}{2}} A \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[I_{|V|} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f}
\end{aligned}
$$

Graph Convolutional Network

- Since eigenvalues of $\boldsymbol{L}$ fall in the interval $[0,2]$, we can approximate $\lambda_{\max} = 2$ and write

$$\begin{aligned}
\mathbf{f} *_G \mathbf{h} &= \theta_0 f + \theta_1 \left[\mathbf{L} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \left(\mathbf{D} - \mathbf{A}\right) \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \mathbf{D} \mathbf{D}^{-\frac{1}{2}} - \mathbf{D}^{-\frac{1}{2}} A \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[I_{|V|} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} - \theta_1 \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}
\end{aligned}$$

Graph Convolutional Network

- Since eigenvalues of $\boldsymbol{L}$ fall in the interval $[0,2]$, we can approximate $\lambda_{\max} = 2$ and write

$$
\begin{aligned}
\mathbf{f} *_G \mathbf{h} &= \theta_0 f + \theta_1 \left[\mathbf{L} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \left(\mathbf{D} - \mathbf{A}\right) \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \mathbf{D} \mathbf{D}^{-\frac{1}{2}} - \mathbf{D}^{-\frac{1}{2}} A \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[I_{|V|} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} - \theta_1 \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}
\end{aligned}
$$

- One can further enforce **parameter sharing** by setting $\theta = \theta_0 = -\theta_1$ and write

Graph Convolutional Network

- Since eigenvalues of $\boldsymbol{L}$ fall in the interval [0,2], we can approximate $\lambda_{\max} = 2$ and write

$$
\begin{aligned}
\mathbf{f} *_G \mathbf{h} &= \theta_0 f + \theta_1 \left[\mathbf{L} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \left(\mathbf{D} - \mathbf{A}\right) \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[\mathbf{D}^{-\frac{1}{2}} \mathbf{D} \mathbf{D}^{-\frac{1}{2}} - \mathbf{D}^{-\frac{1}{2}} A \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} + \theta_1 \left[I_{|V|} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} - I_{|V|}\right] \mathbf{f} \\
&= \theta_0 \mathbf{f} - \theta_1 \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}
\end{aligned}
$$

- One can further enforce **parameter sharing** by setting $\theta = \theta_0 = -\theta_1$ and write

$$
\mathbf{f} *_G \mathbf{h} = \theta \left(I_{|V|} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}\right) \mathbf{f}
$$

Graph Convolutional Network

- Applying $\boldsymbol{K}$ **successive** filters

$$\mathbf{f} *_G \mathbf{h} = \theta \left(I_{|V|} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) \mathbf{f}$$

effectively convolves the $\boldsymbol{k}$**-hop neighborhood** of a node v_i.

Graph Convolutional Network

- Applying $\boldsymbol{K}$ **successive** filters

$$\mathbf{f} *_G \mathbf{h} = \theta \left(I_{|V|} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) \mathbf{f}$$

effectively convolves the $\boldsymbol{k}$**-hop neighborhood** of a node v_i.

- However, since eigenvalues of $I_{|V|} + \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}}$ fall in the interval $[0,2]$, successive application of the operator may be numerically **unstable**.

Graph Convolutional Network

- Applying $\boldsymbol{K}$ **successive** filters

$$\mathbf{f} *_G \mathbf{h} = \theta \left(I_{|V|} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) \mathbf{f}$$

  effectively convolves the $\boldsymbol{k}$**-hop neighborhood** of a node v_i.

- However, since eigenvalues of $I_{|V|} + \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}}$ fall in the interval $[0,2]$, successive application of the operator may be numerically **unstable**.
- Thus, using a **renormalization trick**, one can rewrite this as

$$\mathbf{f} *_G \mathbf{h} = \theta \left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \right) \mathbf{f}$$

  where

$$\tilde{\mathbf{A}} = \mathbf{A} + I_{|V|} \quad \text{and} \quad \widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{ij}$$

Graph Convolutional Network

- For higher dimensional signals $\boldsymbol{H} \in \mathbb{R}^{|V| \times d}$, this results in

$$\boldsymbol{H}^{(t+1)} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \boldsymbol{H}^{(t)} \Theta$$

  Where $\Theta \in \mathbb{R}^{d \times d'}$ and $\boldsymbol{H}^{(t+1)} \in \mathbb{R}^{|V| \times d'}$.

- Adding a **non-linearity** one can arrive at the definition of the **GCN** layer.
- Therefore, **GCN layers** are a **first-order approximation** of the spectral convolution parametrized by Chebyshev polynomials.

Basic GNN

- We observe that a convolution layer defined as a polynomial of $\boldsymbol{I} + \boldsymbol{A}$ is equivalent of
  - **Message aggregation** and
  - **Combining** these information with the information of the node itself.

Basic GNN

- We observe that a convolution layer defined as a polynomial of $\boldsymbol{I} + \boldsymbol{A}$ is equivalent of
  - **Message aggregation** and
  - **Combining** these information with the information of the node itself.
- By adding weights and nan-linearities to this convolution formulation, one can recover the **basic GNN** model

$$\boldsymbol{H}^{(t+1)} = \sigma\left(\mathbf{A}\boldsymbol{F}\Theta_N + \boldsymbol{H}^{(t)}\Theta_v\right)$$

Summary