Graph Neural Networks

Graph Deep Learning 2021 - Lecture 2

Daniele Grattarola

March 1, 2021

Roadmap

Things we are going to cover:

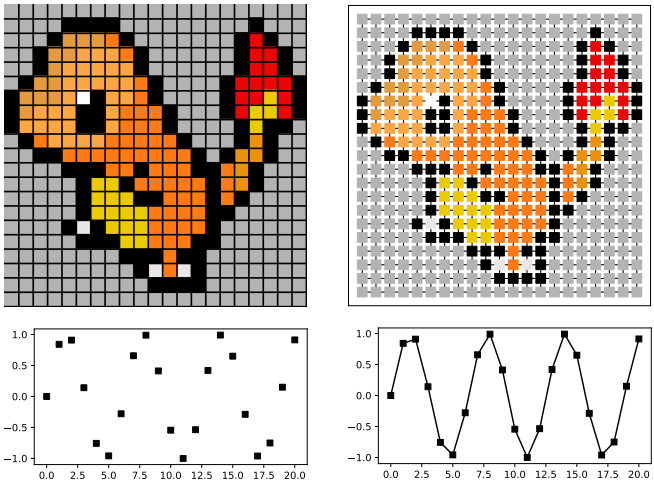
- Practical introduction to GNNs
- Message passing
- Advanced GNNs (attention, edge attributes)
- Demo

After the break:

- Spectral graph theory and GNNs

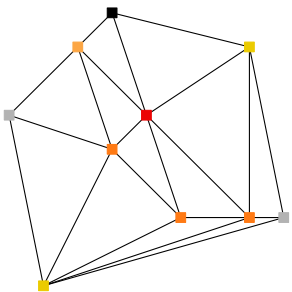
Recall: what are we doing?

From CNNs to GNNs

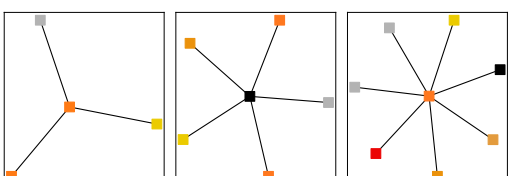


- The receptive field of a CNN reflects the **underlying grid structure**.
- The CNN has an **inductive bias** on how to process the individual pixels/timesteps/nodes.

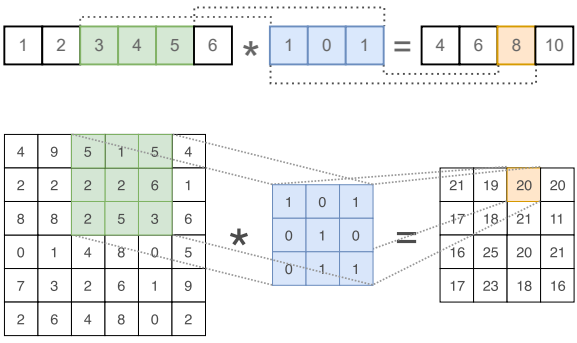
From CNNs to GNNs



- Drop assumptions about underlying structure: it is now an **input of the problem**.
- The only thing we know: the representation of a node depends on its **neighbors**.



Discrete Convolution

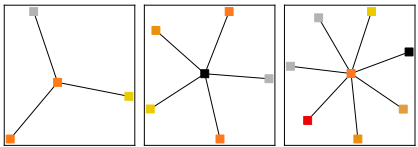


Discrete convolution:

$$(f \star g)[n] = \sum_{m=-M}^{M} f[n-m]g[m]$$

Problems:

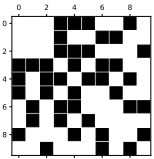
- Variable degree of nodes
- Orientation

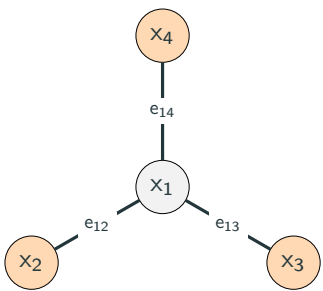


Notation recap

- Graph: nodes connected by edges;
- $\mathrm{X} = [\mathrm{x}_1, \ldots, \mathrm{x}_N]$, $\mathrm{x}_i \in \mathbb{R}^F$, node attributes or "graph signal";
- $\mathrm{e}_{ij} \in \mathbb{R}^S$, edge attribute for edge $i \to j$;
- A, $N \times N$ adjacency matrix;
- $\mathrm{D} = \mathrm{diag}([d_1, \ldots, d_N])$, diagonal degree matrix;
- $\mathrm{L} = \mathrm{D} - \mathrm{A}$, Laplacian;
- $\mathrm{A}_n = \mathrm{D}^{-1/2}\mathrm{A}\mathrm{D}^{-1/2}$, normalized adjacency matrix;
- **Reference operator** R: $\mathrm{r}_{ij} \neq 0$ if $\exists i \to j$

NOTE: all matrices are symmetric.





A quick recipe for a local learnable filter

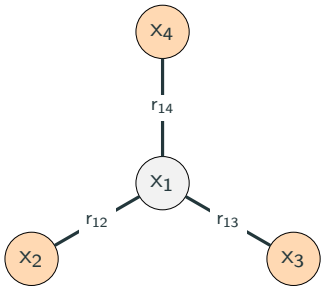
Applying R to graph signal X is a **local** action:

$$(\mathsf{RX})_i = \sum_{j=1}^{N} \mathsf{r}_{ij} \cdot \mathsf{x}_j = \sum_{j \in \mathcal{N}(i)} \mathsf{r}_{ij} \cdot \mathsf{x}_j$$

Instead of having a different weight for each neighbor, **we share weights among nodes in the same neighborhood:**

$$\mathsf{X}' = \mathsf{RX}\Theta$$

where $\Theta \in \mathbb{R}^{F \times F'}$.

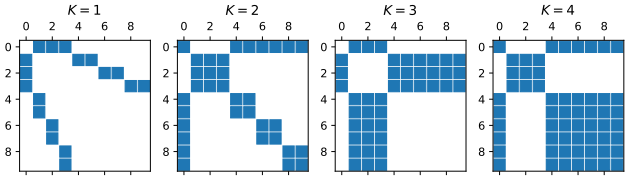


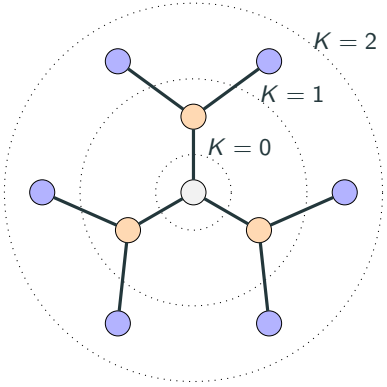
Powers of R

Let's consider the effect of applying R^2 to X:

$$(RRX)_i = \sum_{j \in \mathcal{N}(i)} r_{ij} (RX)_j = \sum_{j \in \mathcal{N}(i)} \sum_{k \in \mathcal{N}(j)} r_{ij} \cdot r_{jk} \cdot x_k$$

Key idea: by applying R^K we **read from the K-th order neighborhood** of a node.

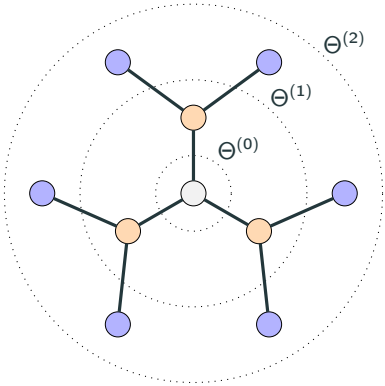




Polynomials of R

To cover all neighbors of order 0 to K, we can just take a polynomial with weights $\Theta^{(k)}$:

$$X' = \sigma\Big(\sum_{k=0}^{K} R^k X \Theta^{(k)}\Big)$$



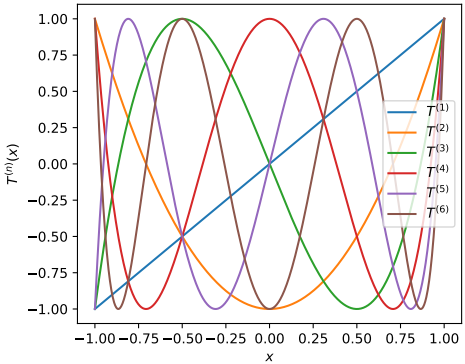
Chebyshev Polynomials [1]

A recursive definition using Chebyshev polynomials:

$$\mathsf{T}^{(0)} = \mathsf{I}$$
$$\mathsf{T}^{(1)} = \tilde{\mathsf{L}}$$
$$\mathsf{T}^{(k)} = 2 \cdot \tilde{\mathsf{L}} \cdot \mathsf{T}^{(k-1)} - \mathsf{T}^{(k-2)}$$

Where $\tilde{\mathsf{L}} = \frac{2\mathsf{L}_n}{\lambda_{\max}} - \mathsf{I}$ and $\mathsf{L}_n = \mathsf{I} - \mathsf{D}^{-1/2}\mathsf{A}\mathsf{D}^{-1/2}$

Layer: $\mathsf{X}' = \sigma\Big(\sum_{k=0}^{K} \mathsf{T}^{(k)}\mathsf{X}\Theta^{(k)}\Big)$



[1] M. Defferrard *et al.*, "Convolutional neural networks on graphs with fast localized spectral filtering," 2016.

Graph Convolutional Networks [2]

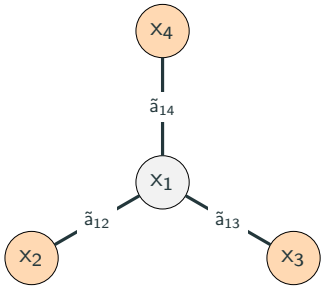
Polynomial of order $K \rightarrow K$ layers of order 1;

Three simplifications:

1. $\lambda_{\max} = 2 \rightarrow \tilde{\mathsf{L}} = \dfrac{2\mathsf{L}_n}{\lambda_{\max}} - \mathsf{I} = -\mathsf{D}^{-1/2}\mathsf{A}\mathsf{D}^{-1/2} = -\mathsf{A}_n$
2. $K = 1 \rightarrow \mathsf{X}' = \mathsf{X}\Theta^{(0)} - \mathsf{A}_n\mathsf{X}\Theta^{(1)}$
3. $\Theta = \Theta^{(0)} = -\Theta^{(1)}$

Layer: $\mathsf{X}' = \sigma\Big(\underbrace{(\mathsf{I} + \mathsf{A}_n)}_{\tilde{\mathsf{A}}}\mathsf{X}\Theta\Big) = \sigma\Big(\tilde{\mathsf{A}}\mathsf{X}\Theta\Big)$

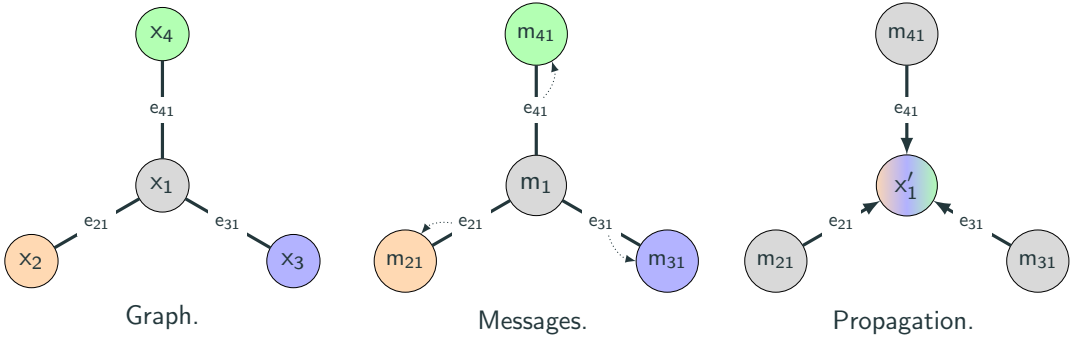
For stability: $\tilde{\mathsf{A}} = \mathsf{D}^{-1/2}(\mathsf{I} + \mathsf{A})\mathsf{D}^{-1/2}$



[2] T. N. Kipf *et al.*, "Semi-supervised classification with graph convolutional networks," 2016.

A General Paradigm

Message Passing Neural Networks [3]



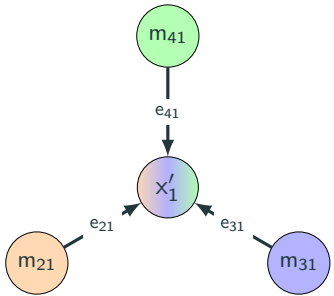
[3] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message Passing Neural Networks [3]

A general scheme for message-passing networks:

$$\mathrm{x}'_i = \gamma\left(\mathrm{x}_i, \square_{j \in \mathcal{N}(i)}\, \phi\left(\mathrm{x}_i, \mathrm{x}_j, \mathrm{e}_{ji}\right)\right),$$

- ϕ: **message function**, depends on x_i, x_j and possibly the edge attribute e_{ji} (we call messages m_{ji});
- $\square_{j \in \mathcal{N}(i)}$: **aggregation function** (sum, average, max, or something else...);
- γ: **update function**, final transformation to obtain new attributes after aggregating messages.



[3] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Models

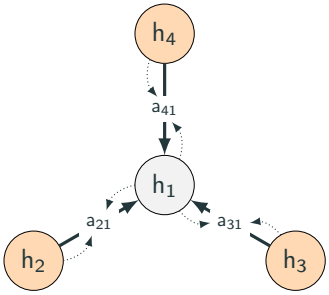
Graph Attention Networks [4]

1. Update the node features: $\mathsf{h}_i = \Theta_f \mathsf{x}_i$ with $\Theta_f \in \mathbb{R}^{F' \times F}$.
2. Compute attention logits: $\mathsf{e}_{ij} = \sigma\left(\theta_a^\top [\mathsf{h}_i \parallel \mathsf{h}_j]\right)$, with $\theta_a \in \mathbb{R}^{2F'}$.[1]
3. Normalize with Softmax:

$$\mathsf{a}_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp\left(\mathsf{e}_{ij}\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(e_{ik}\right)}$$

4. Propagate using the attention coefficients:

$$\mathsf{x}'_i = \sum_{j \in \mathcal{N}(i)} \mathsf{a}_{ij} \mathsf{h}_j$$



[1] $\parallel$ indicates concatenation

[4] P. Velickovic *et al.*, "Graph attention networks," 2017.

Edge-Conditioned Convolution [5]

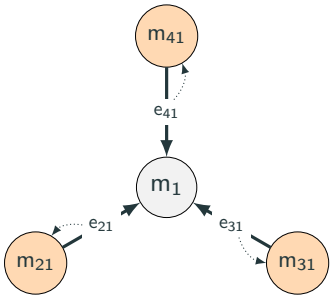
Key idea: incorporate edge attributes into the messages.

Consider a MLP $\phi : \mathbb{R}^S \to \mathbb{R}^{FF'}$ called a **filter generating network**:

$$\Theta^{(ji)} = \text{reshape}(\phi(\mathbf{e}_{ji}))$$

Use the edge-dependent weights to compute messages:

$$\mathbf{x}'_i = \Theta^{(i)}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \Theta^{(ji)}\mathbf{x}_j + \mathbf{b}$$



[5] M. Simonovsky *et al.*, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," 2017.

So which GNN do I use?

GCNConv *Kipf & Welling*	**ChebConv** *Defferrard et al.*	**GraphSageConv** *Hamilton et al.*	**ARMAConv** *Bianchi et al.*
ECCConv *Simonovsky & Komodakis*	**GATConv** *Velickovic et al.*	**GCSConv** *Bianchi et al.*	**APPNPConv** *Klicpera et al.*
GINConv *Xu et al.*	**DiffusionConv** *Li et al.*	**GatedGraphConv** *Li et al.*	**AGNNConv** *Thekumparampil et al.*
TAGConv *Du et al.*	**CrystalConv** *Xie & Grossman*	**EdgeConv** *Wang et al.*	**MessagePassing** *Gilmer et al.*

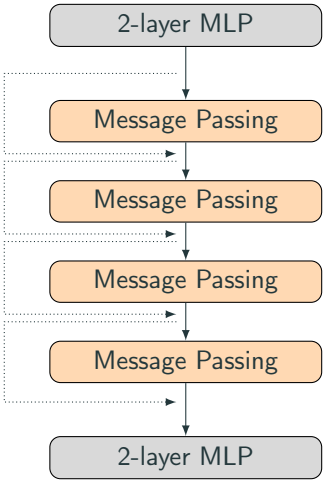
A Good Recipe [6]

Message passing scheme:

- Message: $\mathbf{m}_{ji} = \mathrm{PReLU}\left(\mathrm{BatchNorm}\left(\Theta \mathbf{x}_j + \mathbf{b}\right)\right)$
- Aggregate: $\mathbf{m}^{\mathrm{agg}} = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ji}$
- Update: $\mathbf{x}' = \mathbf{x} \,||\, \mathbf{m}^{\mathrm{agg}}$;

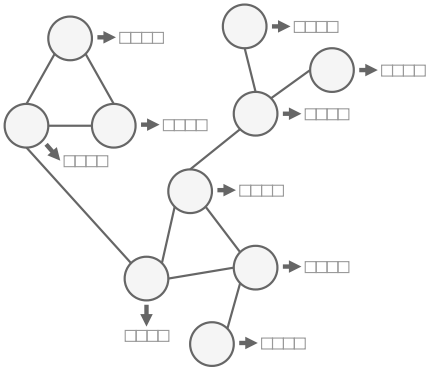
Architecture:

- Pre- and post-process node features using 2-layer MLPs;
- 4-6 message passing steps;

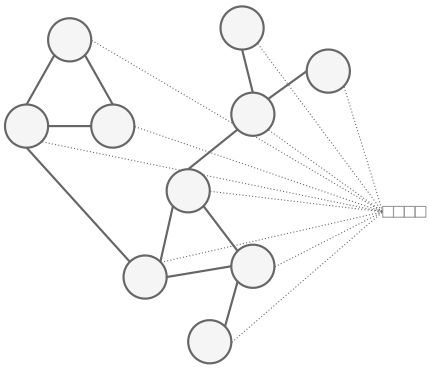


[6] J. You *et al.*, "Design space for graph neural networks," 2020.

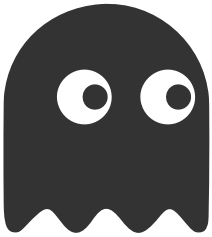
How do we use this?



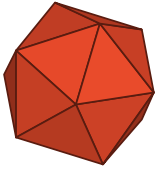
Node-level learning.
(e.g., social networks)



Graph-level learning.
(e.g., molecules)



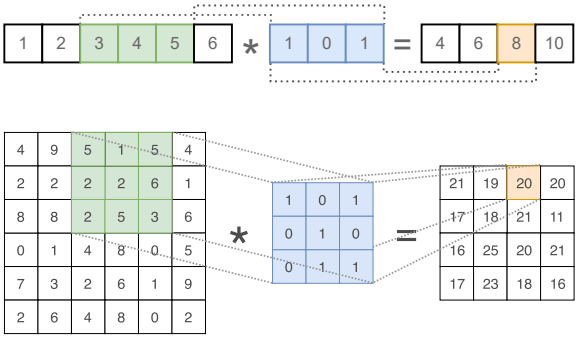
Spektral



PyTorch geometric

Graph Convolution

Discrete Convolution



Recall: CNNs compute a discrete convolution

$$(f \star g)[n] = \sum_{m=-M}^{M} f[n-m]g[m] \quad (1)$$

Convolution Theorem

Given two functions f and g, their convolution $f \star g$ can be expressed as:

$$f \star g = \mathcal{F}^{-1} \{ \mathcal{F} \{ f \} \cdot \mathcal{F} \{ g \} \} \tag{2}$$

Where $\mathcal{F}$ is the **Fourier transform** and $\mathcal{F}^{-1}$ its inverse.

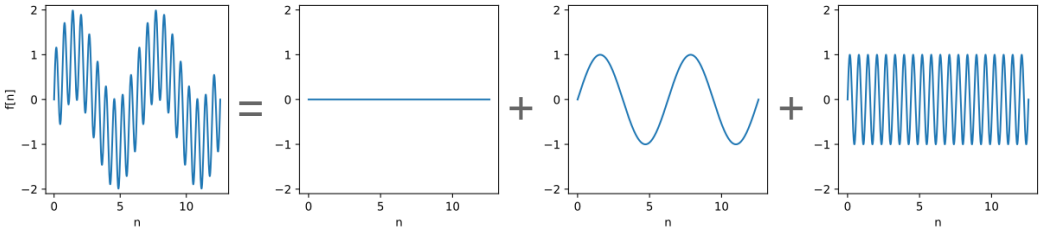
Can we use this major property?

What is the Fourier transform?

Key intuition – we are representing a function in a different basis.

$$\mathcal{F}\{f\}[k] = \hat{f}[k] = \sum_{n=0}^{N-1} f[n] e^{-i\frac{2\pi}{N}kn}$$

$$\mathcal{F}^{-1}\{\hat{f}\}[n] = f[n] = \frac{1}{N}\sum_{k=0}^{N-1} \hat{f}[k] e^{i\frac{2\pi}{N}kn}$$

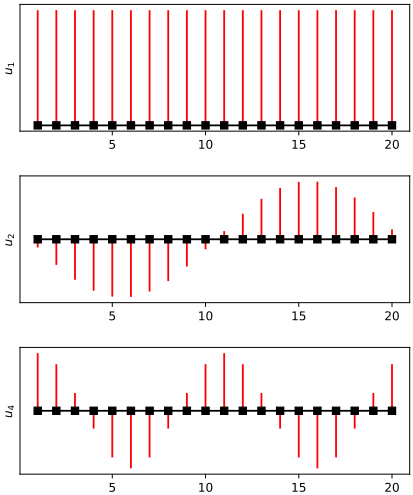


From FT to GFT

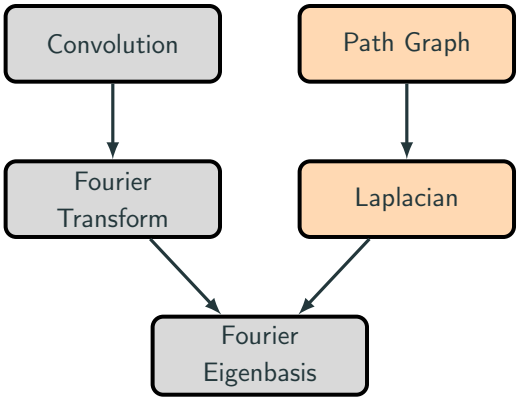
The eigenvectors of the Laplacian for a path graph can be obtained analytically:

$$\mathrm{u}_k[n] = \begin{cases} 1, & \text{for } k = 0 \\ e^{i\pi(k+1)n/N}, & \text{for odd } k, k < N-1 \\ e^{-i\pi kn/N}, & \text{for even } k, k > 0 \\ cos(\pi n), & \text{for odd } k, k = N-1 \end{cases}$$

Looks familiar?



From FT to GFT



- Drop the "grid" assumption
- Replace $e^{-i\frac{2\pi}{N}kn}$ with generic $\mathrm{u}_k[n]$:

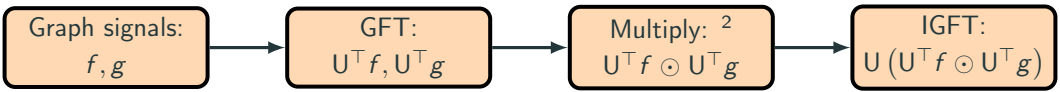
$$\mathcal{F}_G\{f\}[k] = \sum_{n=0}^{N-1} f[n]\mathrm{u}_k[n]$$

- GFT: $\mathcal{F}_G\{f\} = \hat{f} = \mathrm{U}^\top f$;
- IGFT: $\mathcal{F}_G^{-1}\{\hat{f}\} = f = \mathrm{U}\hat{f}$

Graph Convolution

Recall:

- Convolution theorem: $f \star g = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{f\right\} \cdot \mathcal{F}\left\{g\right\}\right\}$
- Spectral theorem: $\mathsf{L} = \mathsf{U}\Lambda\mathsf{U}^\top = \sum_{i=0}^{N-1} \lambda_i \mathsf{u}_i \mathsf{u}_i^\top$



Graph filter: $\mathsf{U}\left(\mathsf{U}^\top f \odot \mathsf{U}^\top g\right) = \mathsf{U} \cdot \underbrace{\mathrm{diag}(\mathsf{U}^\top g)}_{g(\Lambda)} \cdot \mathsf{U}^\top f = \underbrace{\mathsf{U} \cdot g(\Lambda) \cdot \mathsf{U}^\top}_{g(\mathsf{L})} f = g(\mathsf{L}) f$

[2] $\odot$ indicates element-wise multiplication

Spectral GCNs

Spectral GCNs

A first idea [7]: transformation of **each individual eigenvalue** is learned with a free parameter θ_i.

Problems:

- $O(N)$ parameters;
- not **localized** in node space (the only thing that we want);
- $\mathrm{U} \cdot g(\Lambda) \cdot \mathrm{U}^\top$ costs $O(N^2)$;

$$g_\theta(\Lambda) = \begin{bmatrix} \theta_0 & & & & \\ & \theta_1 & & & \\ & & \ddots & & \\ & & & \theta_{N-2} & \\ & & & & \theta_{N-1} \end{bmatrix}$$

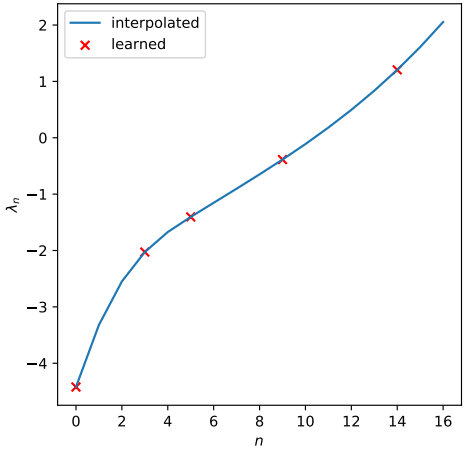
[7] J. Bruna *et al.*, "Spectral networks and locally connected networks on graphs," 2013.

Spectral GCNs

Better idea [7]:

- **Localized** in node domain $\leftrightarrow$ **smooth** in spectral domain;
- Learn only a few parameters θ_i;
- **Interpolate** the other eigenvalues using a smooth **cubic spline**;

Localized and $O(1)$ parameters, but **multiplying by** U **twice is still expensive**.

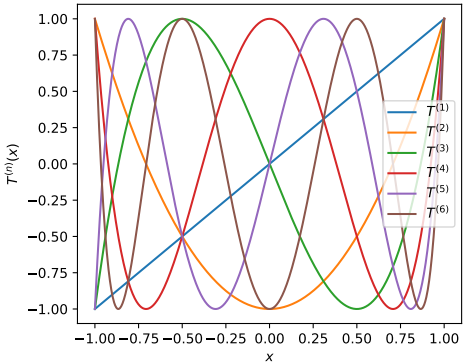


[7] J. Bruna *et al.*, "Spectral networks and locally connected networks on graphs," 2013.

Chebyshev Polynomials [1]

The same recursion is used to filter eigenvalues:

$$\mathsf{T}^{(0)} = \mathsf{I}$$
$$\mathsf{T}^{(1)} = \tilde{\Lambda}$$
$$\mathsf{T}^{(k)} = 2 \cdot \tilde{\Lambda} \cdot \mathsf{T}^{(k-1)} - \mathsf{T}^{(k-2)}$$



[1] M. Defferrard *et al.*, "Convolutional neural networks on graphs with fast localized spectral filtering," 2016.

References i

[1] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2016.

[3] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

[4] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[5] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

References ii

[6] J. You, R. Ying, and J. Leskovec, "Design space for graph neural networks," *arXiv preprint arXiv:2011.08843*, 2020.

[7] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.