Prof. Luana Ruiz

Lecture 8 EN.553.744 DS methods for LS graphs

Today: → popular GNN archits. that can be expressed in conv. form
- MPNN
- GCN
- ChebNet
- GraphSAGE

→ GATs

Recap: GNNs

A GNN is a sequence of L layers of the form:

$$U_\ell = \sum_{k=0}^{K-1} S^k X_{\ell-1} H_{\ell,k}$$

$$X_\ell = \sigma(U_\ell)$$

where: S is the GSO (e.g. $S = A, L, \dots$)

$X_{\ell-1} \in \mathbb{R}^{n \times d_{\ell-1}}$

$H_{\ell,k} \in \mathbb{R}^{d_{\ell-1} \times d_\ell}$

σ: $\mathbb{R} \to \mathbb{R}$ nonlinear (e.g. ReLU)

and: $X_0 = X$; $\quad Y = \Phi_{\mathcal{H}}(X, S) = X_L$

$$\mathcal{H} = \{H_{\ell,k}\}_{\ell,k}$$

This is a convolutional GNN: it is based on convolutional filters (from GSP)

↳ GNNs inherit amazing theoretical properties, which explain their practical success, from them!

But it is general enough to encompass a majority of the most popular architectures in use these days.

- Message-passing neural networks (MPNNs) (Gilmer et al.)

Each layer consists of 2 operations:

1) message: $[m_\ell]_i = \sum_{j \in N(i)} M_\ell\left([x_\ell]_i, [x_\ell]_j, A_{ij}\right)$ $(i \in V)$

2) update: $[x_{\ell+1}]_i = U_\ell\left([x_\ell]_i, [m_\ell]_i\right)$

=> As long as M_ℓ is linear, "message" can be expressed as a graph convolution

e.g.:

$$M_\ell\left([x_\ell]_i, [x_\ell]_j, A_{ij}\right) = \alpha [x_\ell]_i + \beta A_{ij} [x_\ell]_j \quad (j \in N(i))$$

$$\downarrow$$

$m_\ell = \alpha x_\ell + \beta A \cdot x_\ell$ → graph convolution with $K = 2$, $S = A$, $h_0 = \alpha$, $h_1 = \beta$

=> as long as U_ℓ is the composition of a pointwise nonlinearity σ with a linear fcn on $([m_\ell]_i, [x_\ell]_i)$, can be expressed as GNN layer

e.g.: $U_\ell([x_\ell]_i, [m_\ell]_i) = \sigma(\alpha'[x_\ell]_i + \beta'[m_\ell]_i)$

$$x_{\ell+1} = \sigma\left((\alpha' + \beta'\alpha)\, x_\ell + \beta'\beta A x_\ell\right)$$

- σ: pointwise NL
- $(\alpha' + \beta'\alpha)\, x_\ell + \beta'\beta A x_\ell$: graph conv. w/ $S = A$, $K = 2$
  - $h_0 = \alpha' + \beta'\alpha$
  - $h_1 = \beta'\beta$

Ex.: from the perspective of learning the weights (h_0, h_1 vs. $\alpha, \alpha', \beta, \beta'$), what is different?

- Graph convolutional networks (Kipf & Welling, 2017) (GCNs)

A GCN layer is given by:

$$[x_\ell]_i = \sigma\left(\sum_{j \in N(i)} \frac{[x_{\ell-1}]_j}{|N(i)|} H_\ell\right)$$

$$x_\ell = \sigma(S x_{\ell-1} H_\ell)$$

- σ: pointwise nonlinearity
- $S x_{\ell-1} H_\ell$: graph conv. with $K = 2$, $H_{\ell 0} = 0$, $S = D_{bin}^{-\frac{1}{2}} A_{bin} D_{bin}^{-\frac{1}{2}}$, $H_{\ell 1} = H_\ell$

(A_{bin} is the binary adj.
↳ $A_{bin} = (A > 0)$ in Python

↳ advantages:

- only one local diffusion step
- simple / interpretable
- normalization by degree avoids exploding / vanishing gradients

↳ disadvantages:

- only one diffusion step
- no edge weights
- only supports $S = A$
- no "self loops" (unless present in A)
- even if self-loops, no ability to weight $[X_{\ell-1}]_i$ & $[X_{\ell-1}]_j, j \in N(i)$ differently

▶ Chebnets (Defferrard et al., 2017)

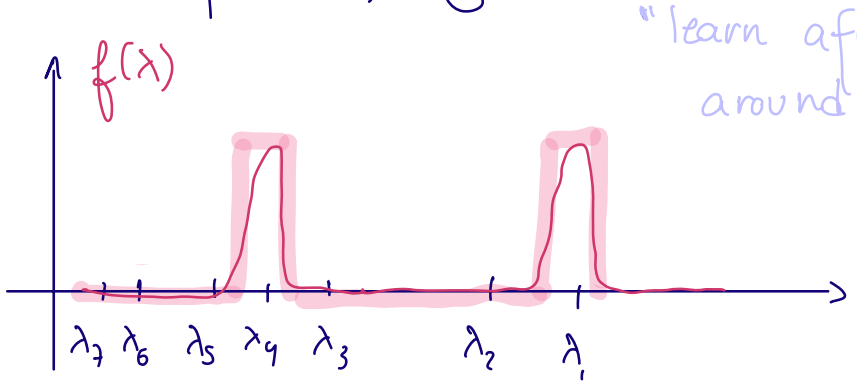
"Fast" implementation of:

$$h(\lambda) = \sum_{k=0}^{K-1} \theta_k \lambda^k(L)$$

↳ Laplacian

which we already know is the spectral domain representation/frequency response of the graph convolution → the weights h_k are the same

Learning in the spectral domain provides an "inductive bias" for learning localized spectral filters, e.g.



However, translating back to the spectral domain is expensive:

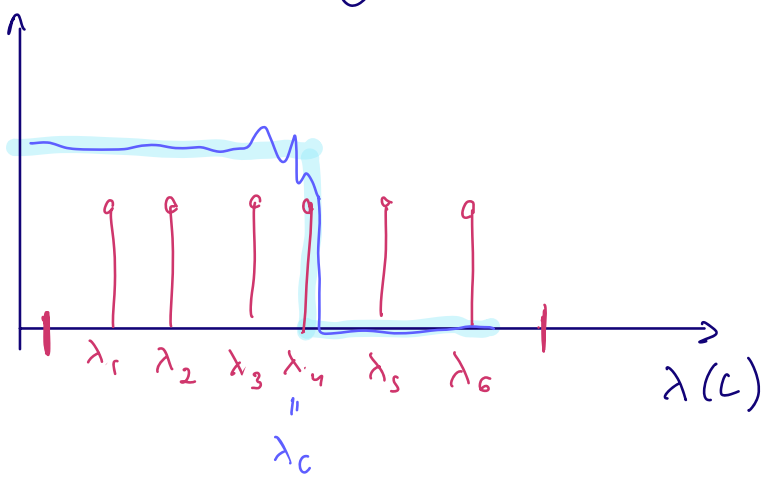
- we need to compute $GFT(x) = \hat{x} = V^T x$ & the inverse GFT of $h(\lambda)\hat{x}$, $y = V h(\lambda)\hat{x}$

↳ cost of 2 additional mv multiplications + cost of eigendecomposition of L

Solution: Chebyshev polynomials. In conventional SP, filters based on these polynomials

have better cutoff behavior in the spectral domain (compared with polynomial filters)

I.e., say we want to approximate the following lowpass filter:



Using $h(\lambda) = \sum_{k=0}^{K-1} h_k \lambda^k$ requires more coeffs.

for the same quality of approx. than the Chebyshev polynomial approx.:

$$h(\lambda) = \sum_{k=0}^{K'-1} h_k T_k(\lambda)$$

i.e., for the same approx. quality, $K' < K$

The Chebyshev polynomials can be efficiently calculated using the following recurrence:

$$\begin{cases} T_0(\lambda) = 1 \\ T_1(\lambda) = \lambda \\ T_{k+1}(\lambda) = 2\lambda T_k(\lambda) - T_{k-1}(\lambda) \end{cases}$$

satisfy $T_n(\cos\theta) = \cos(n\theta)$

Why is Chebyshev expansion better than Taylor expansion in this case?

→ Chebyshev polynomial expansion minimizes L_∞ norm ($\|x\|_\infty = \max_i |x_i|$) over the approximation interval

↳ while Taylor series is a local approximation around same $\tilde{\lambda}$

↳ check the Chebyshev equioscillation theorem for the formal statement & proof.

Further: the polynomials $T_0, T_1, T_2, \ldots$ are orthogonal, so the orthonormal basis comes for free!

↳ with respect to the inner product:

$$\langle f, g \rangle = \int_{-1}^{1} f(x)\, g(x)\, \frac{dx}{\sqrt{1-x^2}}$$

Pf. $\int_{-1}^{1} T_n(x)\, T_m(x) \frac{dx}{\sqrt{1-x^2}}$ $\Rightarrow$ $x = \cos\theta$

$\frac{dx}{d\theta} = \sin\theta$

$\Rightarrow \int_{\pi}^{2\pi} T_n(\cos\theta)\, T_m(\cos\theta) \frac{d\theta \sin\theta}{\sqrt{1-\cos^2\theta}}$ (dx)

$= \int_{\pi}^{2\pi} \cos(n\theta)\cos(m\theta)\, d\theta$ $\quad (T_n = \cos(n\theta))$

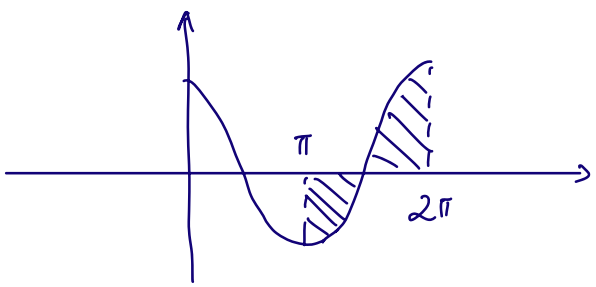
$= \int_{\pi}^{2\pi} \left(\frac{e^{in\theta} + e^{-in\theta}}{2}\right)\left(\frac{e^{im\theta} + e^{-im\theta}}{2}\right) d\theta$

$= \frac{1}{4}\int_{\pi}^{2\pi} e^{-i(n+m)\theta} + e^{-i(n-m)\theta} + e^{-i(m-n)\theta} + e^{i(n+m)\theta}\, d\theta$

a) if $n = m = 0$: $\frac{1}{4}\int_{\pi}^{2\pi} 4\, d\theta = e\Big|_{\pi}^{2\pi} = \boxed{\pi}$

b) if $n = m \neq 0$: $\frac{1}{4}\int_{\pi}^{2\pi} 2 \cdot d\theta + \frac{1}{4}\int_{\pi}^{2\pi} 2 \cdot \cos((n+m)\theta)\, d\theta$

$= \frac{\pi}{2} + 0 = \frac{\pi}{2}$



- if $n \neq m$, then we get

$$\frac{1}{4\pi} \int_{\pi}^{2\pi} 2\cos((m+n)\theta) + 2\cos((n-m)\theta)\, d\theta = \boxed{0}$$

Hence: we get orthogonality for free!

no need to implement in the spectral domain

Hence: we get orthogonality for free!
no need to implement in the spectral domain

Chebnet layer: $X_l = \sigma\left(\sum_{k=0}^{K-1} T_k(\bar{L})\, X_{l-1}\, H_{l,k}\right)$

$\bar{L} = \frac{2L}{\lambda_{max}} - I$ (to ensure spectra btw $(-1, 1]$)

↳ advantages

- fast, cheap
- localized spectral filters

↳ disadvantages

- unstable
- s restricted to L
- difficult to interpret in node domain

- Graph SAGE (Hamilton-Ying-Leskovec, 2017)

Each SAGE layer implements the following operations:

$$[U_\ell]_i = \text{AGGREGATE}_\ell\left(\{[X_{\ell-1}]_j, j \in N(i)\}\right)$$

$$[X_\ell]_i = \sigma\left(\text{CONCAT}([X_{\ell-1}]_i, [U_\ell]_i) H_\ell\right)$$

$$[X_\ell]_i = \frac{[X_\ell]_i}{\|[X_\ell]_i\|_2}$$

The standard AGGREGATE operation is an average over $N(i)$. Letting $H = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}$, we get:

$$[U_\ell]_i = \frac{1}{|N(i)|} \sum_{j \in N(i)} [X_{\ell-1}]_j \Rightarrow U_\ell = S X_{\ell-1}$$

$S = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ ↳ binary

$$X_\ell = \sigma\left([X_{\ell-1} \| U_\ell] \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}\right) = \sigma(X_{\ell-1} H_0 + S X_{\ell-1} H_1)$$

↳ convolutional GNN w/ $K = 2$

↳ equivalent to GCN! the only difference is the node wise normalization $\frac{[X_\ell]_i}{\|[X_\ell]_i\|_2}$, which helps in some cases (empirically)

→ SAGE implementations allow for a variety of AGGREGATE functions, including max, LSTM → no longer convolutional

look at $N(i)$ as a sequence ← ↳ lose perm. equiv.

→ same advantages & disadvantages as GCN

Obs.: the authors of SAGE popularized the AGGREGATE-UPDATE representation of GNNs ($K=2$):

$$X_\ell = \sigma\left(S X_{\ell-1} H_\ell\right)$$

↳ aggregate; → update