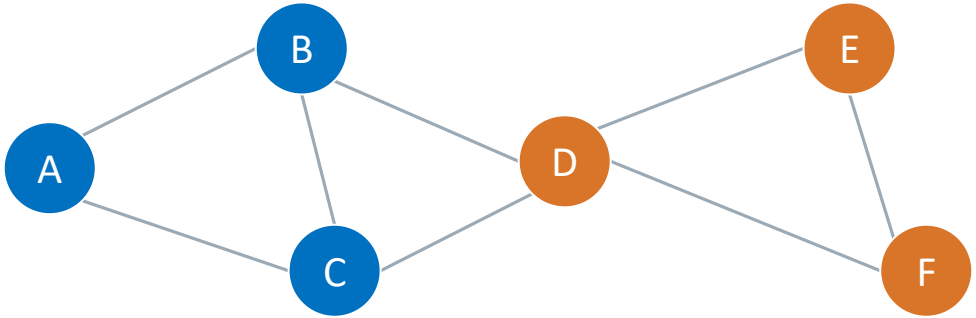
Graph Neural Networks: Part 1

Graphs are structured data

Many real-world datasets describe objects and their relationships.

Social networks



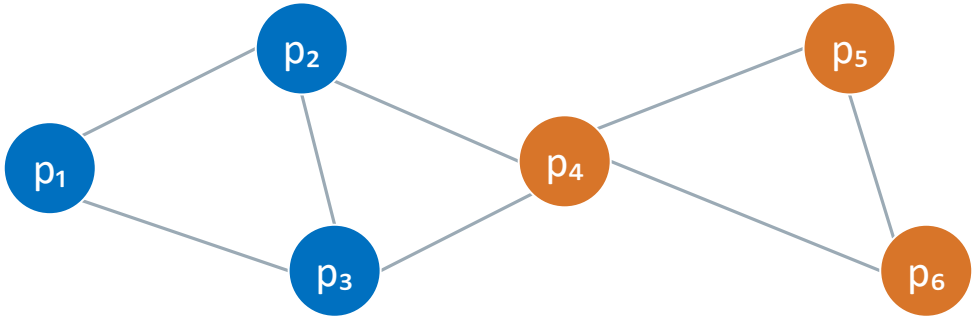
Communities connected by a few friendships

People are nodes. Friendships are edges.

Graphs are structured data

Many real-world datasets describe objects and their relationships.

Social networks • Protein-interaction networks



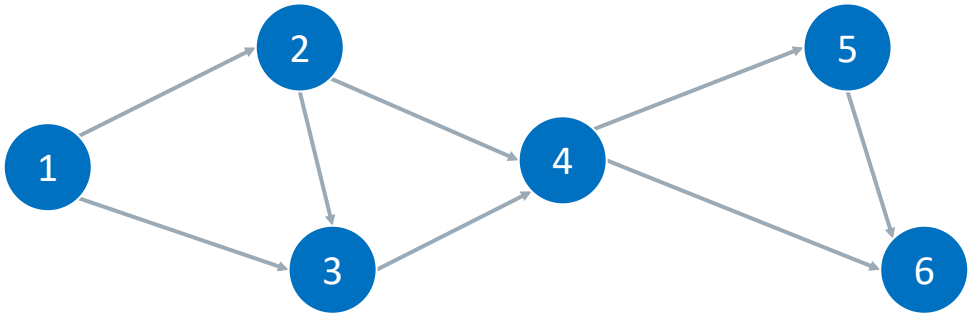
A schematic protein-interaction network

Proteins are nodes. Measured interactions are edges.

Graphs are structured data

Many real-world datasets describe objects and their relationships.

Social networks • Protein-interaction networks • The World Wide Web



Pages connected by hyperlinks

Pages are nodes. Hyperlinks are directed edges.

Graphs are structured data

Many real-world datasets describe objects and their relationships.

Social networks • Protein-interaction networks • The World Wide Web • Molecules

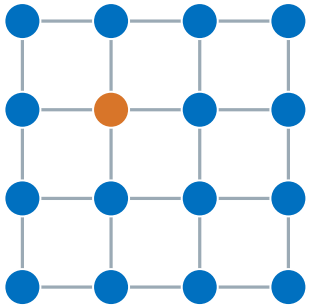


A small molecule: ethanol

Atoms are nodes. Chemical bonds are edges.

Images are graphs

Each pixel is a node. Edges connect adjacent pixels.



Pixel grid

A convolution uses a local neighborhood on this regular grid.

Text as a graph

Each token is a node. Connect each token to the preceding token.



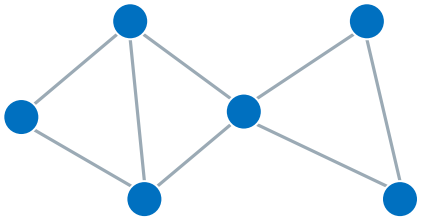
This chain is one possible graph representation of a sentence.

Tasks

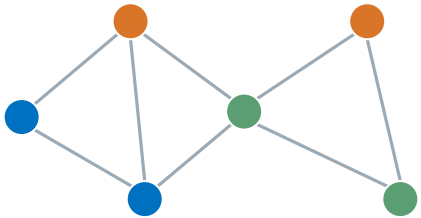
Graph-level task: predict a property of the entire graph.

Node-level task: predict a label for each node.

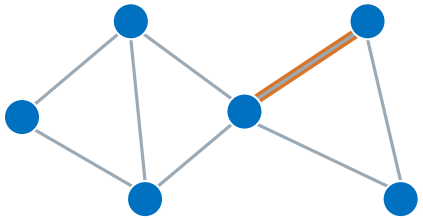
Edge-level task: predict a relationship between two nodes.



Graph label



Node labels



Edge label

Graph-level task

Predict a property of an entire graph.

Example: predict whether a molecule binds to a specified receptor.

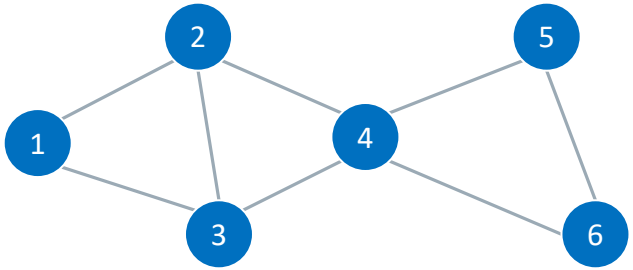


A small molecule: ethanol

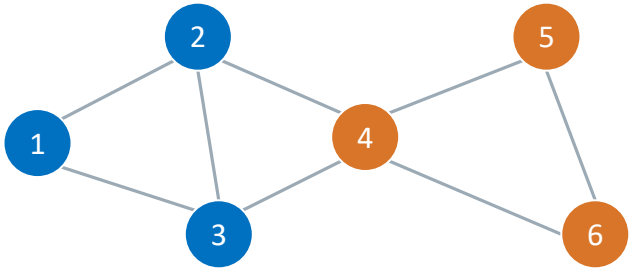
Node and bond information combine into one graph prediction.

Node-level task

Predict the identity or role of individual nodes.



Observed graph

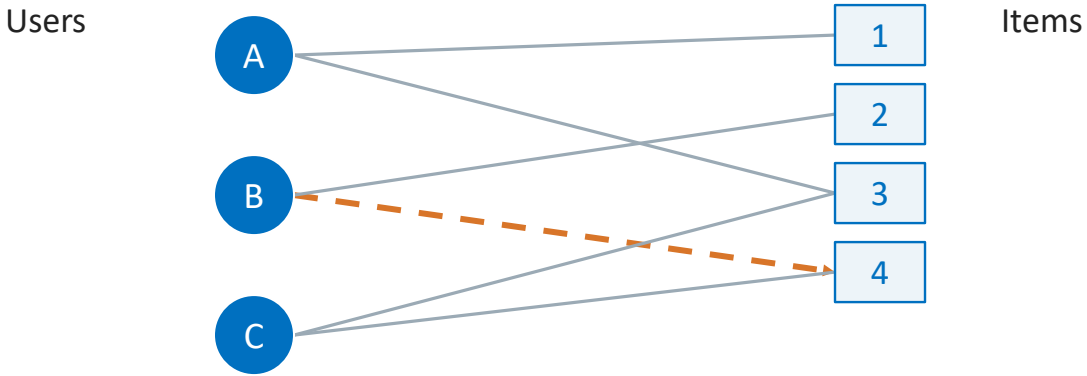


Predicted node classes

Example: predict a paper's topic using its features and citation links.

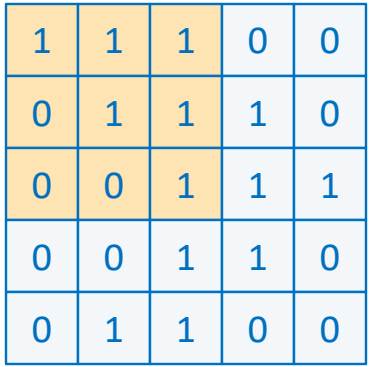
Edge-level task

Predict a missing link or classify an existing relationship.



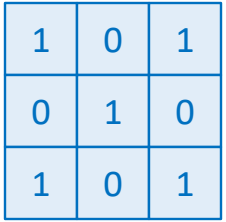
Example: predict which item a user may interact with next.

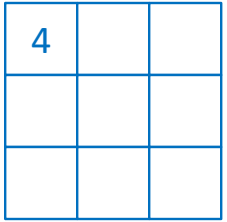
CNN as a GNN



Image

Filter



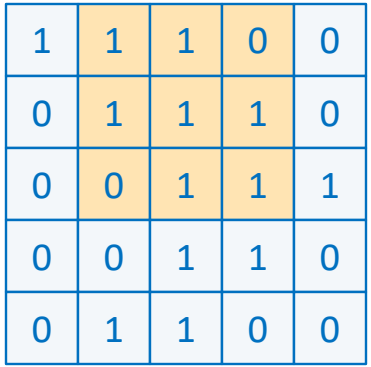


Convolved feature

The same filter moves across local neighborhoods of the pixel grid.

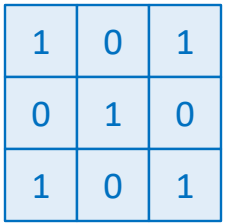
Each output is a weighted sum of nearby pixel values.

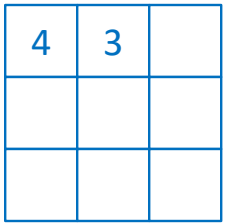
CNN as a GNN



Image

Filter



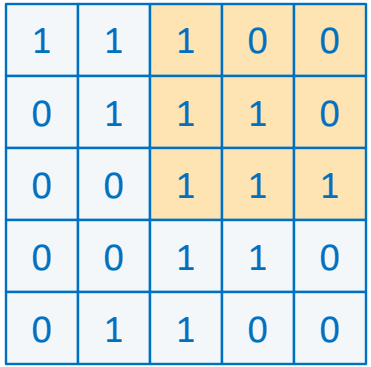


Convolved feature

The same filter moves across local neighborhoods of the pixel grid.

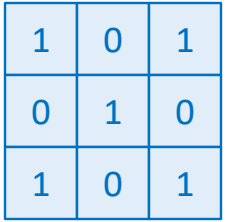
Each output is a weighted sum of nearby pixel values.

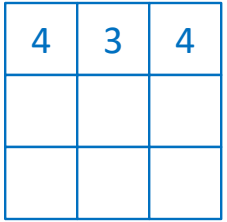
CNN as a GNN



Image

Filter



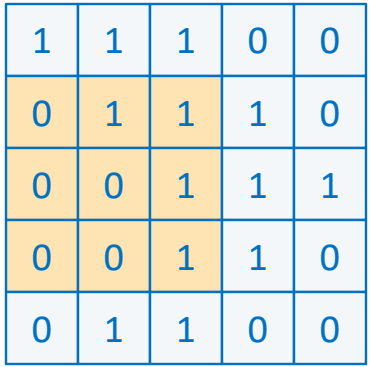


Convolved feature

The same filter moves across local neighborhoods of the pixel grid.

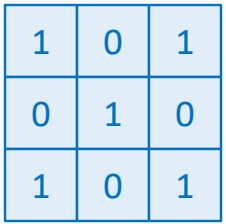
Each output is a weighted sum of nearby pixel values.

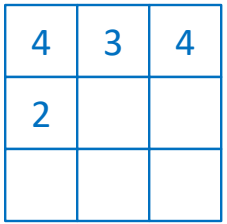
CNN as a GNN



Image

Filter



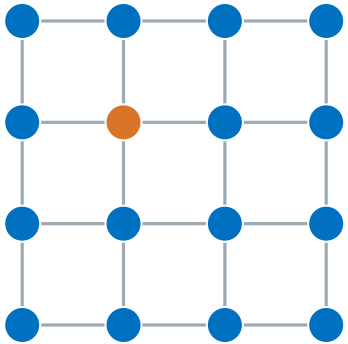


Convolved feature

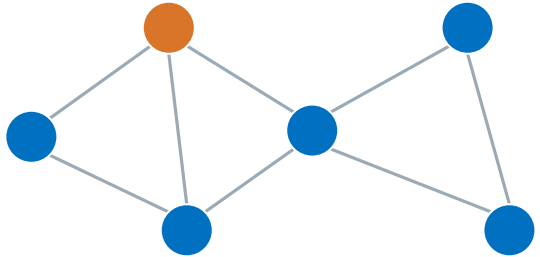
The same filter moves across local neighborhoods of the pixel grid.

Each output is a weighted sum of nearby pixel values.

Regular and irregular neighborhoods



Pixel grid



Irregular graph

An image has a fixed grid. A graph can have different numbers of neighbors.

Convolution

A convolution combines an input signal with a shifted filter.

$$s(t) = \int_{-\infty}^{\infty} x(a)w(t-a)\,da$$

We use an asterisk to denote this operation.

$$s(t) = (x * w)(t)$$

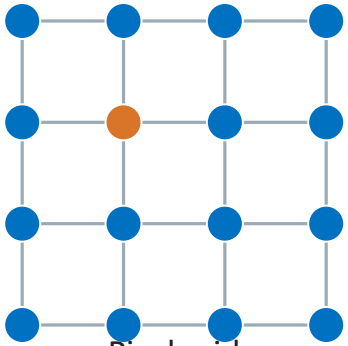
Discrete convolution

For signals defined at integer positions, replace the integral with a sum.

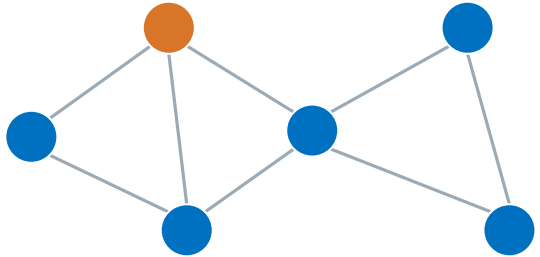
$$s[t] = (x * w)[t] = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

The filter weights are shared across positions.

Convolution on graphs



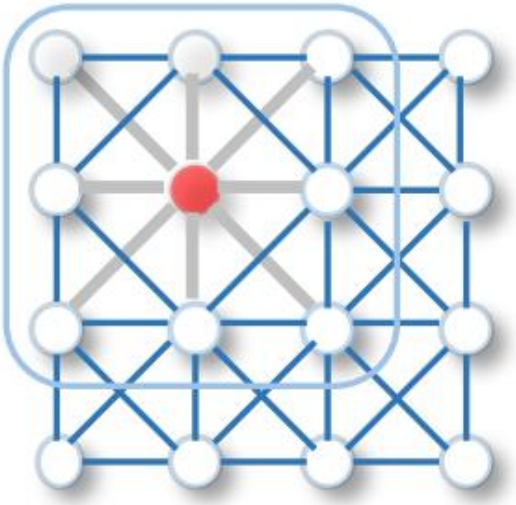
Pixel grid

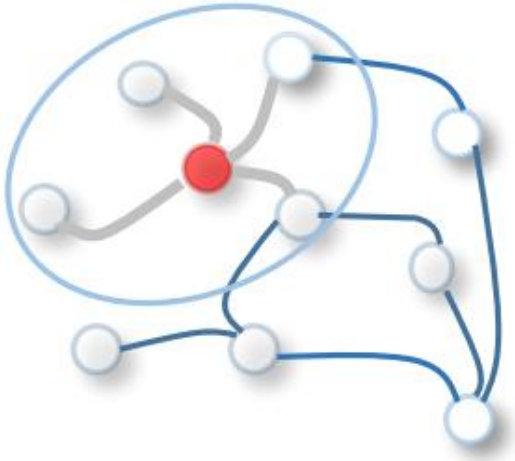


Irregular graph

How can we define a shared filter when there is no regular grid?

We first use the graph Laplacian to define a Fourier basis.





Definition of a graph

A graph consists of vertices, edges, and an adjacency matrix.

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, A), \qquad |\mathcal{V}| = n, \quad A \in \mathbb{R}^{n \times n}$$

$$A_{ij} = \begin{cases} 1, & \text{if nodes } i \text{ and } j \text{ are connected,} \\ 0, & \text{otherwise.} \end{cases}$$

For the spectral derivation, assume an undirected graph with nonnegative weights.

$$A = A^{\mathsf{T}}, \qquad A_{ij} \geq 0$$

Laplacian of a graph

The Laplacian records differences between a node and its neighbors.

$$L = D - A$$

The degree matrix places the weighted node degrees on its diagonal.

$$D = \mathrm{diag}(d_1, \ldots, d_n), \qquad d_i = \sum_{j=1}^{n} A_{ij}$$

Degree matrix and degree of a vertex

The degree is the sum of the weights in one adjacency row.

$$d_i = \sum_{j=1}^{n} A_{ij}$$

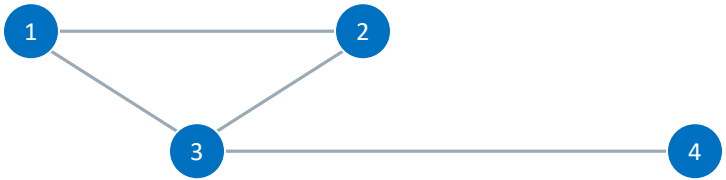
$$D = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{bmatrix}$$

For an unweighted graph without self-loops, this counts the neighbors.

Example

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \qquad D = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$L = D - A = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$



Graph cut optimization

A cut separates nodes while keeping few edges between the groups.

$$\mathbf{y}^\top L\mathbf{y} = \frac{1}{2}\sum_{i,j} A_{ij}(y_i - y_j)^2$$

A spectral relaxation finds a smooth, nonconstant assignment:

$$\min_{\mathbf{y}} \ \mathbf{y}^\top L\mathbf{y} \quad \text{subject to} \quad \mathbf{y}^\top\mathbf{y} = 1, \quad \mathbf{y}^\top\mathbf{1} = 0$$

For a connected graph, a second eigenvector solves this relaxation.

Thresholding it gives a partition, not a general guarantee of the optimal discrete cut.

Eigendecomposition of the Laplacian

$$L = U \Lambda U^\mathsf{T}$$

$$U = [\mathbf{u}_1, \ldots, \mathbf{u}_n], \qquad U^\mathsf{T} U = I$$

$$\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n), \qquad L\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

The eigenvectors form an orthonormal basis.

Each eigenvalue describes the variation of its eigenvector across edges.

Normalized Laplacian

Start with the ordinary Laplacian and normalize by node degree.

$$L = D - A$$

$$L_{\text{sym}} = D^{-1/2}(D - A)D^{-1/2}$$

$$L_{\text{sym}} = I - D^{-1/2}AD^{-1/2}$$

Assume positive degrees here. Self-loops later ensure this condition.

We reserve a tilde for the scaled Laplacian used in ChebNet.

Laplacian eigenvectors and Fourier analysis

Fourier analysis expresses a signal as a sum of sinusoidal components.

Sines and cosines form an orthogonal basis, ordered by frequency.

On a graph, Laplacian eigenvectors play the role of these basis functions.

$$\mathbf{x} = \sum_{i=1}^{n} \widehat{x}_i \mathbf{u}_i$$

Eigenvectors of the Laplacian

Small eigenvalues: neighboring nodes tend to have similar values.

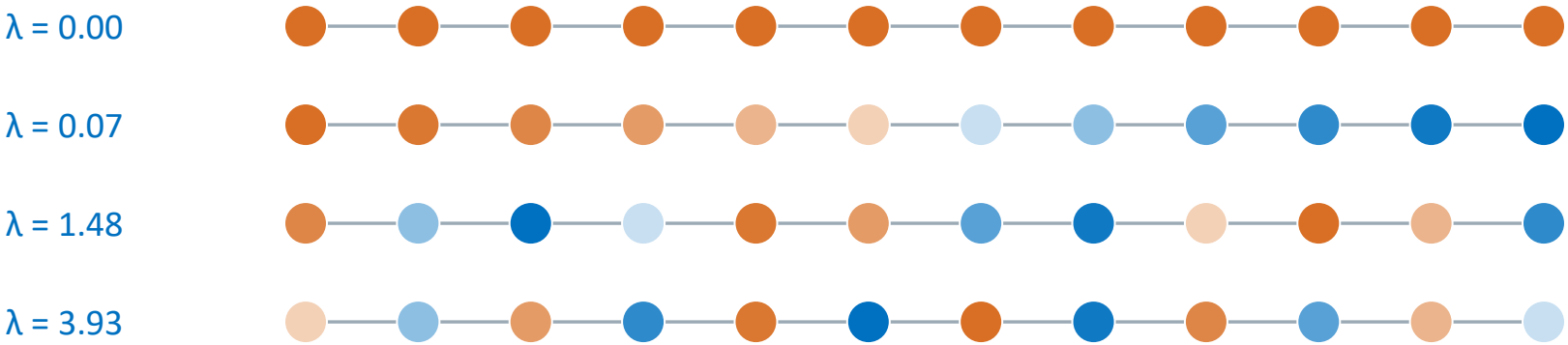
Large eigenvalues: values can change rapidly across connected nodes.

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$$

Ordering by eigenvalue gives a graph analogue of frequency.

Graph frequency on a path

The same path graph supports slowly varying and rapidly varying signals.



Node colors show eigenvector values. More sign changes indicate higher frequency.

Fourier functions

$$L = U\Lambda U^{\mathsf{T}}, \qquad U = [\mathbf{u}_1, \ldots, \mathbf{u}_n]$$

$$\Lambda_{ii} = \lambda_i, \qquad U^{\mathsf{T}}U = I$$

The eigenvectors are the graph Fourier basis functions.

The diagonal entries of the eigenvalue matrix form the spectrum.

Fourier transform

Project the graph signal onto each Fourier basis function.

$$\widehat{x}_i = \mathbf{u}_i^\mathsf{T}\mathbf{x}$$

The resulting coefficients describe how much of each frequency is present.

$$\widehat{\mathbf{x}} = U^\mathsf{T}\mathbf{x}$$

Fourier transform

A graph signal contains one scalar value for every node.

$$\mathbf{x} \in \mathbb{R}^n$$

Forward transform:

$$\mathcal{F}(\mathbf{x}) = U^\mathsf{T}\mathbf{x} = \widehat{\mathbf{x}}$$

Inverse transform:

$$\mathcal{F}^{-1}(\widehat{\mathbf{x}}) = U\widehat{\mathbf{x}} = UU^\mathsf{T}\mathbf{x} = \mathbf{x}$$

Convolution theorem

Define graph convolution by multiplying Fourier coefficients elementwise.

$$\mathcal{F}(\mathbf{x} * \mathbf{g}) = \mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g})$$

$$\mathbf{x} * \mathbf{g} = U\left[(U^{\mathsf{T}}\mathbf{x}) \odot (U^{\mathsf{T}}\mathbf{g})\right]$$

$$= U\,\mathrm{diag}(U^{\mathsf{T}}\mathbf{g})U^{\mathsf{T}}\mathbf{x} = UG_{\boldsymbol{\theta}}U^{\mathsf{T}}\mathbf{x}$$

$$U^{\mathsf{T}}g = \begin{bmatrix} u_g^{\mathsf{T}} \\ u_{ng}^{\mathsf{T}} \end{bmatrix}$$

$$G_\theta = \begin{bmatrix} u_1^{\mathsf{T}}g & \cdots & \square \\ \vdots & \ddots & \vdots \\ \square & \cdots & u_n^{\mathsf{T}}g \end{bmatrix}$$

The diagonal filter matrix contains one multiplier per graph frequency.

Vanilla Spectral GCN and ChebNet

Convolutional graph neural networks

- **Spectral methods**

define filters through the graph Fourier basis.

- **Spatial methods**

define updates by passing information between nodes.

Polynomial spectral filters connect these two viewpoints.

GCN uses a local propagation rule motivated by a first-order spectral approximation.

Convolution theorem

A spectral filter scales Fourier coefficients, then transforms back.

$$\mathcal{F}(\mathbf{x} * \mathbf{g}) = \mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g})$$

$$\mathbf{x} * \mathbf{g} = U\left[(U^{\mathsf{T}}\mathbf{x}) \odot (U^{\mathsf{T}}\mathbf{g})\right]$$

$$= U\,\mathrm{diag}(U^{\mathsf{T}}\mathbf{g})U^{\mathsf{T}}\mathbf{x} = UG_{\boldsymbol{\theta}}U^{\mathsf{T}}\mathbf{x}$$

$$G_{\boldsymbol{\theta}} = \mathrm{diag}(U^{\mathsf{T}}\mathbf{g})$$

Neural network layers

A feedforward layer transforms a feature vector:

$$\mathbf{h}' = \sigma(W\mathbf{h})$$

Each layer's output becomes the next layer's input.

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

A convolutional layer uses a shared filter:

$$\mathbf{h}' = \sigma(\mathbf{w} * \mathbf{h})$$

Vanilla spectral GNN

With one node per column, apply the spectral filter on the right.

$$H \in \mathbb{R}^{d \times n}, \quad H' = \sigma\big(HUG_{\boldsymbol{\theta}}U^{\mathsf{T}}\big)$$

$$X = H^{(0)}, \qquad G_{\boldsymbol{\theta}} = \operatorname{diag}(\theta_1, \ldots, \theta_n)$$

The diagonal entries are learnable frequency multipliers.

A feature transformation can additionally mix the channels.

Limitation

A full dense eigendecomposition is expensive on a large graph.

$$\text{Dense eigendecomposition: } \mathcal{O}(n^3)$$

An unrestricted spectral filter learns a separate multiplier for every frequency.

ChebNet avoids explicitly computing the full eigenbasis.

ChebNet

Approximate the spectral filter using Chebyshev polynomials.

$$G_{\boldsymbol{\theta}} = \sum_{i=0}^{k} \theta_i \, t_i(\widetilde{\Lambda})$$

The polynomial degree controls the spatial reach of the filter.

We write the scalar Chebyshev polynomial with a lowercase letter.

Chebyshev Polynomials of the First Kind

$$\cos(\theta) = \cos(\theta)$$
$$\cos(2\theta) = 2\cos^2(\theta) - 1$$
$$\cos(3\theta) = 4\cos^3(\theta) - 3\cos(\theta)$$

Chebyshev polynomials

Substituting $x = \cos(\theta)$ gives:

$$T_1(x) = x$$
$$T_2(x) = 2x^2 - 1$$
$$T_3(x) = 4x^3 - 3x$$

$T_n(x)$ is the polynomial of degree n.

Orthogonal on $[-1, 1]$

with weight $w(x) = 1 / \sqrt{(1 - x^2)}$.

Chebyshev Polynomials of the First Kind

$$T_0(x) = 1$$

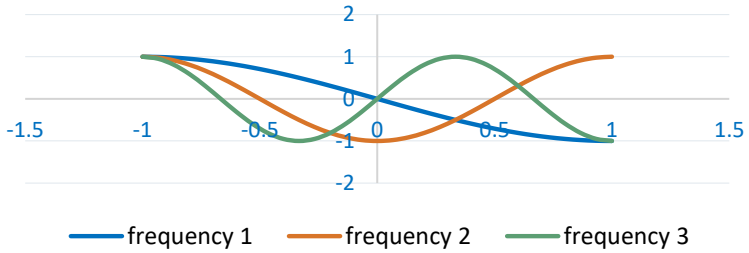
$$T_1(x) = x$$

For $i \geq 2$, $T_i(x) = 2xT_{i-1}(x) - T_{i-2}(x)$

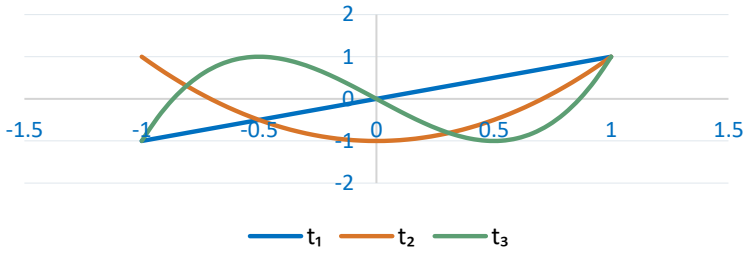
Fourier functions and Chebyshev polynomials

Both are families of basis functions, but their roles here differ.

Fourier-like cosine functions



Chebyshev polynomials



Fourier functions describe a signal over nodes.

Chebyshev polynomials approximate the filter response over eigenvalues.

$$g_{\boldsymbol{\theta}}(\lambda) = \sum_{i=0}^{k} \theta_i \, t_i(\widetilde{\lambda})$$

ChebNet graph convolution

Scale the eigenvalues into the Chebyshev interval:

$$\widetilde{\Lambda} = \frac{2\Lambda}{\lambda_{\max}} - I, \qquad -1 \le \widetilde{\lambda}_j \le 1$$

$$G_{\boldsymbol{\theta}} = \sum_{i=0}^{k} \theta_i \, t_i(\widetilde{\Lambda})$$

$$\mathbf{x} * \mathbf{g}_{\boldsymbol{\theta}} = U G_{\boldsymbol{\theta}} U^{\mathsf{T}} \mathbf{x}$$

$$= \sum_{i=0}^{k} \theta_i \, U t_i(\widetilde{\Lambda}) U^{\mathsf{T}} \mathbf{x}$$

ChebNet graph convolution

$$\mathbf{x} * \mathbf{g}_{\boldsymbol{\theta}} = U G_{\boldsymbol{\theta}} U^{\mathsf{T}} \mathbf{x}$$

$$= U \left(\sum_{i=0}^{k} \theta_i \, t_i(\widetilde{\Lambda}) \right) U^{\mathsf{T}} \mathbf{x}$$

$$= \sum_{i=0}^{k} \theta_i \, U t_i(\widetilde{\Lambda}) U^{\mathsf{T}} \mathbf{x}$$

$$= \sum_{i=0}^{k} \theta_i \, t_i(\widetilde{L}) \mathbf{x}$$

A polynomial of a diagonalized matrix can be evaluated directly on the matrix.

ChebNet graph convolution

$$\mathbf{x} * \mathbf{g}_{\boldsymbol{\theta}} = U G_{\boldsymbol{\theta}} U^{\mathsf{T}} \mathbf{x}$$

$$= U \left(\sum_{i=0}^{k} \theta_i \, t_i(\widetilde{\Lambda}) \right) U^{\mathsf{T}} \mathbf{x}$$

$$= \sum_{i=0}^{k} \theta_i \, U t_i(\widetilde{\Lambda}) U^{\mathsf{T}} \mathbf{x}$$

$$= \sum_{i=0}^{k} \theta_i \, t_i(\widetilde{L}) \mathbf{x}$$

$$\widetilde{L} = \frac{2L}{\lambda_{\max}} - I$$

Evaluate the polynomial without a full eigendecomposition.

Why the polynomial identity works

$$\widetilde{L} = U\widetilde{\Lambda}U^{\mathsf{T}}$$

$$\widetilde{L}^2 = U\widetilde{\Lambda}\underbrace{U^{\mathsf{T}}U}_{I}\widetilde{\Lambda}U^{\mathsf{T}} = U\widetilde{\Lambda}^2U^{\mathsf{T}}$$

$$\widetilde{L}^r = U\widetilde{\Lambda}^rU^{\mathsf{T}}$$

Apply this identity to every power in a polynomial:

$$t_i(\widetilde{L}) = Ut_i(\widetilde{\Lambda})U^{\mathsf{T}}$$

Computing ChebNet with the recurrence

$$\mathbf{z}_0 = \mathbf{x}, \qquad \mathbf{z}_1 = \widetilde{L}\mathbf{x}$$

$$\mathbf{z}_i = 2\widetilde{L}\mathbf{z}_{i-1} - \mathbf{z}_{i-2}$$

$$\mathbf{x} * \mathbf{g}_{\boldsymbol{\theta}} = \sum_{i=0}^{k} \theta_i \mathbf{z}_i$$

Each multiplication uses only a node and its neighbors.

A degree-k polynomial reaches at most k hops.

Lecture 19