Graph Deep Learning

Fabian Bosshard

January 31, 2026

Contents

1 Spectral theorem

Theorem 1.1. Let $\underline{\boldsymbol{A}} \in \mathbb{R}^{n\times n}$ be symmetric, i.e. $\underline{\boldsymbol{A}}^\top = \underline{\boldsymbol{A}}$. Then,

$$\underline{\boldsymbol{A}} = \sum_{i=1}^{n} \lambda_i \, \boldsymbol{u}_i \boldsymbol{u}_i^\top \tag{1.1}$$

with orthonormal eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n \in \mathbb{R}^n$ and real eigenvalues $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$. Equivalently,

$$\underline{\boldsymbol{A}} = \underline{\boldsymbol{U}}\underline{\boldsymbol{\Lambda}}\underline{\boldsymbol{U}}^\top \tag{1.2}$$

with $\underline{\boldsymbol{U}} := \begin{bmatrix}\boldsymbol{u}_1 \cdots \boldsymbol{u}_n\end{bmatrix}$, $\underline{\boldsymbol{U}}^\top \underline{\boldsymbol{U}} = \underline{\boldsymbol{I}}_n$ and $\underline{\boldsymbol{\Lambda}} := \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. $\triangleleft$

Theorem 1.1 is extensively used in Principal Component Analysis (PCA) to reduce the complexity of the input space (it is applied to the covariance matrix of the inputs).

2 Graphs

Let $G = (V, E)$ be an undirected graph with the adjacency matrix $\underline{\boldsymbol{A}} \in \mathbb{R}^{n \times n}$

$$[\underline{\boldsymbol{A}}]_{uv} = \begin{cases} 1 & (u,v) \in E \\ 0 & (u,v) \notin E \end{cases} \tag{2.1}$$

where $[\cdot]_{uv}$ denotes the entry in row u and column v.

The *diagonal degree matrix* $\underline{\boldsymbol{D}} \in \mathbb{R}^{n \times n}$ is defined by

$$[\underline{\boldsymbol{D}}]_{uv} = \begin{cases} d_u & u = v \\ 0 & u \neq v \end{cases} \tag{2.2}$$

where d_u is the degree of node u, i.e. $\underline{\boldsymbol{D}}$ simply places all node degrees on the diagonal.

2.1 normalized adjacency and multi-hop propagation

Definition 2.1. The *symmetrically normalized adjacency matrix* is

$$\boxed{\hat{\underline{\boldsymbol{A}}} = \underline{\boldsymbol{D}}^{-1/2} \underline{\boldsymbol{A}} \underline{\boldsymbol{D}}^{-1/2}} \tag{2.3}$$

or, entrywise,

$$[\hat{\underline{\boldsymbol{A}}}]_{uv} = \begin{cases} \dfrac{1}{\sqrt{d_u d_v}} & (u,v) \in E \\ 0 & (u,v) \notin E \end{cases}$$

$\blacktriangleleft$

Fact 2.1 (multi-hop propagation). The entry $(\hat{\underline{\boldsymbol{A}}}^k)_{vu}$ can be computed explicitly as follows:

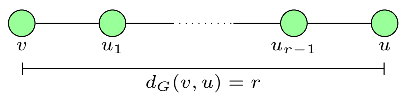
$$[\hat{\underline{\boldsymbol{A}}}^k]_{vu} = \sum_{\pi} \prod_{(x,y) \in E_\pi} \frac{1}{\sqrt{d_x d_y}} \tag{2.4}$$

the sum is over all walks $\pi = (v, \ldots, u)$ of length k from v to u and the product is over the edges $E_\pi = \{(v, u_1), \ldots, (u_{k-1}, u)\}$ on the walk. $\triangleleft$

Corollary 2.2. Let $v, u \in V$ with $r = d_G(v,u)$, where $d_G(\cdot,\cdot)$ denotes the shortest-path distance. Assume there is exactly one path

$$(v, u_1, \ldots, u_{r-1}, u)$$

of length r between v and u:



Then

$$(\hat{\underline{\boldsymbol{A}}}^r)_{vu} = \frac{1}{\sqrt{d_v d_{u_1}}} \cdot \prod_{i=1}^{r-2} \frac{1}{\sqrt{d_{u_i} d_{u_{i+1}}}} \cdot \frac{1}{\sqrt{d_{u_{r-1}} d_u}} = \frac{1}{\sqrt{d_v d_u}} \prod_{i=1}^{r-1} \frac{1}{d_{u_i}} \tag{2.5}$$

$\triangleleft$

2.2 graph Laplacian

Definition 2.2. The *combinatorial* graph Laplacian is

$$\boxed{\underline{\boldsymbol{L}} = \underline{\boldsymbol{D}} - \underline{\boldsymbol{A}}} \tag{2.6}$$

and the *normalized* graph Laplacian is

$$\boxed{\hat{\underline{\boldsymbol{L}}} = \underline{\boldsymbol{D}}^{-1/2} \underline{\boldsymbol{L}} \underline{\boldsymbol{D}}^{-1/2} = \underline{\boldsymbol{D}}^{-1/2} (\underline{\boldsymbol{D}} - \underline{\boldsymbol{A}}) \underline{\boldsymbol{D}}^{-1/2} \overset{(2.3)}{=} \underline{\boldsymbol{I}}_n - \hat{\underline{\boldsymbol{A}}}} \tag{2.7}$$

Both are symmetric and positive semidefinite, and their eigenvalues satisfy

$$0 = \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1}$$

λ_1 is called the *spectral gap*. The number of zero eigenvalues (i.e., the multiplicity of the 0 eigenvalue) equals the number of connected components of the graph. $\blacktriangleleft$

To understand Definition 2.2, consider a function $f\colon V \to \mathbb{R}$. Denote by $\boldsymbol{f} \in \mathbb{R}^n$ the vector whose v-th entry is $f(v)$. Then

$$(\hat{\underline{\boldsymbol{L}}}\boldsymbol{f})_v = f(v) - \frac{1}{\sqrt{d_v}} \sum_{(u,v)\in E} \frac{f(u)}{\sqrt{d_u}} \tag{2.8}$$

i.e., $(\hat{\underline{\boldsymbol{L}}}\boldsymbol{f})_v$ is the value at v minus a degree-normalized average of the neighbors. This is why the Laplacian is often viewed as a *discrete second derivative* on the graph: it measures how much f at v deviates from its neighborhood.

If we plot the eigenvectors of the Laplacian, it resembles that of a signal.

Another important identity is the quadratic form

$$\boldsymbol{f}^\top \underline{\boldsymbol{L}} \boldsymbol{f} = \frac{1}{2} \sum_{(u,v)\in E} \big(f(u) - f(v)\big)^2 \tag{2.9}$$

which shows that $\underline{\boldsymbol{L}}$ (and hence also $\hat{\underline{\boldsymbol{L}}}$) is positive semidefinite, since the right-hand side is always nonnegative. Moreover, (2.9) is small exactly when f varies slowly across edges, so the Laplacian encodes the *smoothness* of functions on the graph.

2.3 Cheeger inequality

The *Cheeger inequality* relates the spectral gap λ_1 to the *Cheeger constant* $h(G)$, which measures how difficult it is to separate the graph into two large pieces. It states, in particular, that

$$\tfrac{1}{2}h(G)^2 \le \lambda_1 \le 2h(G),$$

so a larger spectral gap implies that the graph is more "well-connected".

2.4 effective resistance

Definition 2.3 (effective resistance)**.** View each edge $(u,v) \in E$ as an electrical resistor of resistance $1\,\Omega$. The resulting network has a well-defined resistance between any two nodes.

For two nodes $s,t \in V$, the *effective resistance* $R(s,t)$ is defined as the voltage difference needed to send one unit of electrical current from s to t. It can be computed as

$$\boxed{R(s,t) = (\boldsymbol{e}_s - \boldsymbol{e}_t)^\top \underline{\boldsymbol{L}}^\dagger (\boldsymbol{e}_s - \boldsymbol{e}_t)} \tag{2.10}$$

where $\underline{\boldsymbol{L}}^\dagger$ is the Moore–Penrose pseudoinverse of the graph Laplacian (2.6) and $\boldsymbol{e}_v$ is the standard basis vector of vertex v. ◀

2.4.1 Interpretation

If the graph offers many short, parallel paths between s and t, then current can flow easily, so $R(s,t)$ is small. If there are few or long paths, the current is "bottlenecked" and $R(s,t)$ is large. Thus, effective resistance measures how "well-connected" two nodes are inside the global geometry of the graph.

2.4.2 Connection to random walks

A *random walk* on G is the Markov chain that, from a node v, moves to a uniformly random neighbor of v. Its transition matrix is

$$\boxed{\underline{\boldsymbol{P}} = \underline{\boldsymbol{D}}^{-1}\underline{\boldsymbol{A}}} \tag{2.11}$$

so $\underline{\boldsymbol{P}}_{vu} = 1/d_v$ if $(v,u) \in E$. The matrix (2.11) is often called random-walk matrix.

For two nodes u,v, the *commute time* $\mathtt{CT}(u,v)$ is the expected number of steps for the random walk to start at u, reach v, and return to u again. It can be related to the effective resistance via

$$\mathtt{CT}(u,v) = 2|E|R(u,v) \tag{2.12}$$

giving a geometric interpretation of how "far apart" two nodes are in terms of random-walk behavior, i.e. two nodes have small commute time exactly when they have small effective resistance.

3 Graph neural networks

3.1 Graph shift operators

Definition 3.1. A matrix $\underline{\tilde{\boldsymbol{A}}} \in \mathbb{R}^{n\times n}$ is called a *graph shift operator* (GSO) if it satisfies

$$\boxed{\tilde{a}_{ij} = 0 \quad \text{whenever } (i,j) \notin E \text{ and } i \neq j}$$

where $\tilde{a}_{ij} = [\underline{\tilde{\boldsymbol{A}}}]_{ij}$. This means that applying $\underline{\tilde{\boldsymbol{A}}}$ to node attributes only mixes information from direct neighbors. Typical choices include the Laplacian (2.6) and the random-walk matrix (2.11). ◀

Applying $\underline{\tilde{\boldsymbol{A}}}$ to node attributes $\underline{\boldsymbol{X}} \in \mathbb{R}^{n\times d_x}$ is local in the sense that the i-th row of $\underline{\tilde{\boldsymbol{A}}}\underline{\boldsymbol{X}}$ depends only the neighbors $N(i)$ together with possibly i itself:

$$\boldsymbol{x}_i' = [\underline{\tilde{\boldsymbol{A}}}\underline{\boldsymbol{X}}]_{i,:} = \sum_{j=1}^{n} \tilde{a}_{ij}\boldsymbol{x}_j = \tilde{a}_{ii}\boldsymbol{x}_i + \sum_{j\in N(i)} \tilde{a}_{ij}\boldsymbol{x}_j$$

Using a parameter matrix $\underline{\boldsymbol{\Theta}} \in \mathbb{R}^{d_x\times d_h}$, we can apply the filter on a different space:

$$\boldsymbol{h}_i = [\underline{\tilde{\boldsymbol{A}}}\underline{\boldsymbol{X}}\underline{\boldsymbol{\Theta}}]_{i,:} = \sum_{j=1}^{n} \tilde{a}_{ij}\boldsymbol{x}_j\underline{\boldsymbol{\Theta}} = \tilde{a}_{ii}\boldsymbol{x}_i\underline{\boldsymbol{\Theta}} + \sum_{j\in N(i)} \tilde{a}_{ij}\boldsymbol{x}_j\underline{\boldsymbol{\Theta}}$$

3.2 Graph convolutions

Let $\underline{\boldsymbol{X}} \in \mathbb{R}^{n\times d_x}$ be node features, let $\underline{\tilde{\boldsymbol{A}}} \in \mathbb{R}^{n\times n}$ be a GSO and let $\underline{\boldsymbol{\Theta}} \in \mathbb{R}^{d_x\times d_h}$ be trainable weights. A *linear graph convolution* is

$$\underline{\boldsymbol{H}} = \underline{\tilde{\boldsymbol{A}}}\underline{\boldsymbol{X}}\underline{\boldsymbol{\Theta}} \tag{3.1}$$

where $\underline{\boldsymbol{H}} \in \mathbb{R}^{n\times d_h}$ are the transformed node features.

Adding a nonlinearity σ yields a *graph convolutional layer*

$$\boxed{\underline{\boldsymbol{H}} = \sigma(\underline{\tilde{\boldsymbol{A}}}\underline{\boldsymbol{X}}\underline{\boldsymbol{\Theta}})} \tag{3.2}$$

so the parameters can be learned by gradient-based optimization.

Remark 3.1 (multi-hop aggregation)**.** Stacking K layers increases the receptive field. Ignoring nonlinearities for intuition, applying two layers gives $\underline{\tilde{\boldsymbol{A}}}(\underline{\tilde{\boldsymbol{A}}}\underline{\boldsymbol{X}}) = \underline{\tilde{\boldsymbol{A}}}^2\underline{\boldsymbol{X}}$ which aggregates information from 2-hop neighborhoods. ◀

Two common ways to aggregate up to K hops are polynomial filters

$$\underline{\boldsymbol{H}}^{(K)} = \sum_{k=0}^{K} \underline{\tilde{\boldsymbol{A}}}^k \underline{\boldsymbol{X}}\underline{\boldsymbol{\Theta}}^{(k)} \tag{3.3}$$

or a sequence of first-order steps $\underline{\boldsymbol{H}}^{(0)} = \underline{\boldsymbol{X}}$, $\underline{\boldsymbol{H}}^{(k)} = \underline{\tilde{\boldsymbol{A}}}\underline{\boldsymbol{H}}^{(k-1)}\underline{\boldsymbol{\Theta}}^{(k)}$ where nonlinearities can be inserted between layers.

3.3 Examples of choices for $\underline{\tilde{\boldsymbol{A}}}$

Several popular layers differ essentially by the choice of $\underline{\tilde{\boldsymbol{A}}}$. A standard example is the *GCN normalization*

$$\boxed{\underline{\tilde{\boldsymbol{A}}} = \underline{\boldsymbol{D}}^{-1/2}(\underline{\boldsymbol{I}}_n + \underline{\boldsymbol{A}})\underline{\boldsymbol{D}}^{-1/2}} \tag{3.4}$$

which includes self-loops through $\underline{\boldsymbol{I}}_n + \underline{\boldsymbol{A}}$. Another example is the random-walk normalization (2.11) which corresponds to averaging over neighbors with probabilities. A third example is the *GIN* choice

$$\underline{\tilde{\boldsymbol{A}}} = \underline{\boldsymbol{A}} + (1+\varepsilon)\underline{\boldsymbol{I}}_n \tag{3.5}$$

which strengthens the contribution of the root node via ε.

3.4 Message passing

Definition 3.2. Let $\boldsymbol{x}_i \in \mathbb{R}^{d_x}$ be the feature of node i and let $\boldsymbol{e}_{ji} \in \mathbb{R}^{d_e}$ be the feature of edge (j, i). A *message-passing* (MP) layer has the form

$$\boxed{\boldsymbol{h}_i = \gamma\Big(\boldsymbol{x}_i,\, \mathsf{agg}_{j\in N(i)}\big\{\phi(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{e}_{ji})\big\}\Big)} \tag{3.6}$$

where

- ϕ is a message function, depending on $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ and possibly edge features $\boldsymbol{e}_{ji}$
- agg is a permutation-invariant aggregation function (e.g. sum, mean, max)
- γ is an update function to obtain new features from aggregated messages and previous features

Note that ϕ and γ are often parametric (e.g. MLPs). ◀

Remark 3.2. Definition 3.2 is the most general (and expressive) form of GNN, encompassing also graph convolutions (3.2) as a special case. (3.2) can be rewritten as

$$\boldsymbol{h}_i = \sigma\Big(\sum\nolimits_{j\in N(i)} a_{ij}\boldsymbol{x}_j\underline{\boldsymbol{\Theta}}\Big)$$

with $a_{ij} = \tilde{\underline{\boldsymbol{A}}}_{ij}$. So it fits into Definition 3.2 with $\phi(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{e}_{ji}) = a_{ij}\boldsymbol{x}_j\underline{\boldsymbol{\Theta}}$ (independent of $\boldsymbol{x}_i$ and $\boldsymbol{e}_{ji}$), $\mathsf{agg} = \text{sum}$, and $\gamma(\boldsymbol{x}_i, \cdot) = \sigma(\cdot)$ (independent of $\boldsymbol{x}_i$). ◀

Message passing operations whose message function ϕ depends only on the sender node's features are called *isotropic*. They are called *anisotropic* when also edge's or receiver node's features are exploited, i.e. ϕ also depends on $\boldsymbol{e}_{ji}$ or $\boldsymbol{x}_i$.

3.4.1 Graph attention networks

are a typical example of anisotropic message passing.

1. Transform node features:
$$\boldsymbol{x}_i' = \boldsymbol{x}_i\underline{\boldsymbol{\Theta}}_1 \tag{3.7}$$
with $\underline{\boldsymbol{\Theta}}_1 \in \mathbb{R}^{d_x\times d_h}$.
2. Compute attention scores between neighbors:
   2.1. Score ($\boldsymbol{\theta}_2 \in \mathbb{R}^{2d_h}$):
$$\alpha_{ij} = \sigma\big([\boldsymbol{x}_i' \parallel \boldsymbol{x}_j']\boldsymbol{\theta}_2\big) \tag{3.8}$$
   2.2. Normalize with softmax over $N(i)$:
$$\tilde{\alpha}_{ij} = \frac{\exp(\alpha_{ij})}{\sum_{k\in N(i)} \exp(\alpha_{ik})} \tag{3.9}$$
3. Aggregate (weighted sum) using attention coefficients as weights:
$$\boldsymbol{h}_i = \sum_{j\in N(i)} \tilde{\alpha}_{ij}\,\boldsymbol{x}_j' \tag{3.10}$$

3.4.2 Edge-conditioned convolution

To incorporate edge attributes into the messages, one may use an MLP $\rho\colon \mathbb{R}^{d_e} \to \mathbb{R}^{d_x\times d_h}$ to generate edge-dependent weights. For each edge (j, i) we compute

$$\underline{\boldsymbol{\Theta}}_{ji} = \rho(\boldsymbol{e}_{ji}) \in \mathbb{R}^{d_x\times d_h} \tag{3.11}$$

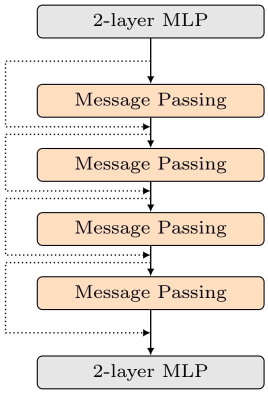
and update nodes by

$$\boldsymbol{h}_i = \boldsymbol{x}_i\underline{\boldsymbol{\Theta}}_i + \sum_{j\in N(i)} \boldsymbol{x}_j\underline{\boldsymbol{\Theta}}_{ji} \tag{3.12}$$

so edges directly control how neighbor information is transformed.

3.5 A good recipe

is to pre- and post-process node features with a 2-layer MLP and to use 4 to 6 message-passing steps in between:



Message passing at the ℓ-th layer:

1. Message:
$$\boldsymbol{m}_{ji}^{\ell} = \mathrm{PReLU}\Big(\mathrm{BatchNorm}\big(\boldsymbol{h}_j^{\ell}\underline{\boldsymbol{\Theta}}^{\ell} + \boldsymbol{b}^{\ell}\big)\Big) \tag{3.13}$$

2. Aggregate by summation:
$$\boldsymbol{m}_i^{\ell} = \sum_{j \in N(i)} \boldsymbol{m}_{ji}^{\ell} \tag{3.14}$$

3. Update by concatenation:
$$\boldsymbol{h}_i^{\ell+1} = \boldsymbol{h}_i^{\ell} \,\|\, \boldsymbol{m}_i^{\ell} \tag{3.15}$$

Remark 3.3. This message-passing instance is *isotropic*, since the message (3.13) depends only on the sender embedding $\boldsymbol{h}_j^{\ell}$ and not on edge attributes or the receiver features. ◀

3.6 Over-smoothing

Repeated graph convolutions tend to reduce feature differences across neighbors, behaving like low-pass filtering on the graph. After many layers, node representations can become almost indistinguishable within connected components, which can harm node-level prediction tasks.

4 Pooling on graphs

Pooling builds coarser graphs to reduce size or to obtain hierarchical representations.

4.1 SRC decomposition

1. **S**election: A selection operator computes K supernodes
$$\mathrm{SEL}\colon G \mapsto S = \{S_1, \dots, S_K\}$$
where each S_k is a set of nodes equipped with nonnegative scores. Equivalently, selection can be encoded by a matrix $\underline{\boldsymbol{S}} \in \mathbb{R}^{K\times n}$.

2. **R**eduction: Given a selection matrix $\underline{\boldsymbol{S}} \in \mathbb{R}^{K\times n}$ and node features $\underline{\boldsymbol{X}} \in \mathbb{R}^{n\times d_x}$, a typical reduction is the weighted aggregation
$$\underline{\boldsymbol{X}}' = \underline{\boldsymbol{S}}\,\underline{\boldsymbol{X}}$$
which produces pooled features $\underline{\boldsymbol{X}}' \in \mathbb{R}^{K\times d_x}$.

3. **C**onnection: A common connection rule builds the pooled adjacency by aggregating edges between supernodes. Using the same $\underline{\boldsymbol{S}}$, a standard choice is
$$\underline{\boldsymbol{A}}' = \underline{\boldsymbol{S}}\,\underline{\boldsymbol{A}}\,\underline{\boldsymbol{S}}^\top$$
which yields $\underline{\boldsymbol{A}}' \in \mathbb{R}^{K\times K}$.

Remark 4.1 (spectral intuition)**.** Low-frequency eigenvectors of the Laplacian reveal coarse clusters. A classical pipeline performs clustering (e.g. k-means) in the space spanned by the first few Laplacian eigenvectors, but this can be expensive and ignores attributes. ◀

4.2 Global pooling

For graph-level tasks, one often needs a graph-to-vector map. A *global pooling* (or *readout*) aggregates node embeddings $\{\boldsymbol{h}_i\}_{i\in V}$ into a single vector and must be permutation-invariant. Typical choices include sum, mean or max pooling, as well as attention-weighted sums.