What's a Graph Neural Network?

Samuel Vaiter

2024-02-15

Contents

There is a lot of expository materials on Graph Neural Networks (GNNs) out there. I want here to focus on a short "mathematical" introduction to it, in the sense to quickly arrive to the concept of equivariance and invariance to permutations, and how GNNs are designed to deal with it, in order to discuss in a future post our work (??, ????) on the convergence of GNNs to some "continuous" limit when the number of nodes goes to infinity.

1 Graphs

Graphs are ubiquitous structure [1] in mathematics and computer science. Graphs may represent several interactions, such as brain connectivity, computer graphic meshes, protein interactions or social networks. In its most elementary definition, a **graph** is a couple $G = (V, E)$ where V is a finite set of **vertices** or **nodes** $V = \{v_1, \dots, v_n\}$, and E is a subset of the *unordered* pairs $\mathcal{P}_2(V)$ of V, called **edges** $E = \{e_{i_1 j_1}, \dots, e_{i_m j_m}\} \subseteq \mathcal{P}_2(V)$.

width=.9]intro-to-gnn-bipartite-graph.svg

Figure 1: A bipartite graph.

Beyond this formalism, it is quite frequent to enumerate the vertices $1, \dots, n$ and consider the associated **adjacency matrix** $A \in \mathbb{R}^{n \times n}$ defined by $A = (a_{i,j})_{1 \leq i,j \leq n}$ where

$$a_{i,j} = \begin{cases} 1 & \text{if } i \sim j, \\ 0 & \text{otherwise.} \end{cases}$$

[1] I think at this point this sentence is a meme in machine learning. At least, it is one for me.

where $i \sim j$ means that there is an edge between i and j. Note that A is a real symmetric matrix.

The graph in Figure **??** can be represented by the adjacency matrix

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

A popular way to encode it concretely on a computer is to use the so-called **adjacency list**, that we can represent mathematically speaking as a couple $G = (V, \mathcal{N})$ where V is a finite set and $\mathcal{N} : V \to \mathcal{P}_2(V)$ is the neighbor function such $j \in \mathcal{N}(i)$ if, and only if, $\{i, j\} \in E$.

The graph in Figure **??** can be represented by the adjacency list

$$1 \mapsto \{4, 5\}, 2 \mapsto \{5, 6\}, 3 \mapsto \{4, 6\},$$
$$4 \mapsto \{1, 3\}, 5 \mapsto \{1, 2\}, 6 \mapsto \{2, 3\}.$$

We also say that a vertex i has a **degree** d_i of k when it has exactly k neighbors. Alternatively, d_i is defined as the cardinal of $\mathcal{N}(i)$.

Note that when speaking about graphs, one might also think of more general structures such as

- Digraphs (directed graphs) where E is not a subset of $\mathcal{P}_2(V)$ but a subset of $V \times V$;
- Graphs with self-loop, i.e., $E \subseteq \mathcal{P}_2(V) \cup V$;
- Multi/Hyper-graph where E might have repeated edges;
- or quivers, i.e., multi-digraphs, etc.

Another important generalization is the consideration of **weighted** (di)graph where instead of dealing with a couple $G = (V, E)$, we add a (*symmetric* if undirected) function $\omega : V \times V \to \mathbb{R}_{>0}$. It is then common to use the weighted adjacency matrix A^ω and weighted degree d_i^ω, defined as

$$a_{i,j}^\omega = \begin{cases} \omega(i,j) & \text{if } i \sim j, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{and } d_i^\omega = \sum_{j \in \mathcal{N}(i)} \omega(\omega_{i,j}).$$

Note that we recover the standard definition of a unweighted graph if $\omega_{i,j} = 1$ when $i \sim j$.

2 Graphs in supervised learning

As a reminder, we typically say that we are performing supervised learning when we have access to a *training set*

$$\mathcal{T} = \{x_i, y_i\}_{i=1}^n \in (\mathcal{X} \times \mathcal{Y}) \sim \mu \quad \text{i.i.d.},$$

where μ is a probability measure over $\mathcal{X} \times \mathcal{Y}$. The goal is to learn from it a function $\Phi : \mathcal{X} \to \mathcal{Y}$ that *generalize* well in the sense that for $(x, y) \sim \mu$, $\Phi(x) \approx y$. This is typically – but not only – achieved using Empirical Risk Minimization (ERM), that is solving the minimizing problem

$$\arg\min_{\Phi \in \mathfrak{F}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, \Phi(x_i)),$$

where $\mathcal{L} : \mathcal{Y} \times \mathcal{Y}$ is an adequate loss function, e.g, the ℓ^2 loss $\mathcal{L}(y, y') = \|y - y'\|_2^2$, and $\mathfrak{F}$ is a candidate set of function $\mathfrak{F} \subseteq \mathcal{Y}^{\mathcal{X}}$. Modern (deep) learning is often considered in the *interpolating regime*, i.e., we seek highly parameterized models where the Φ is expected to achieve a 0-error on the training set: $\forall i \in \{1, \dots, n\}$, $\Phi(x_i) = y_i$.

2.1 Graph classification

Classification of graphs aims to answer questions such as "*Does this protein G is useful (+1) or not (-1)?*" Formally, it is an application (for binary classification)

$$\bar{\Phi} : \begin{cases} \mathfrak{G} & \to \{-1, +1\} \\ G & \mapsto \Phi(G) \end{cases}$$

where $\mathfrak{G}$ is the set of all graphs with a finite number of vertices. Here, the object of interest is the *domain*, e.g, the graph. To learn this function $\bar{\Phi}$, we typically have access to a training set $(G_i, y_i)_{i=1}^n \in (\mathfrak{G} \times \{-1, 1\})^n$.

2.2 Node classification

In contrast, node classification aims to answer "*Does this atom i of charge $z(i)$ involved (+1) or not (-1) in the activity of the protein G?*" Here, the map of interest has the form

$$\Phi : \begin{cases} \mathcal{H}(V) & \to \{-1, +1\} \\ z & \mapsto \Phi(z) \end{cases}$$

where $\mathcal{H}(V)$ is the set of all functions $z : V \to \mathbb{R}$, or equivalently, **node features** $z \in \mathbb{R}^{|V|}$. To learn this function Φ, the training set has the form $(z_i, y_i)_{i=1}^n \in (\mathcal{H}(V) \times \{-1, 1\})^n$. If both task are built around the idea of a graph G, they *fundamentally differ* as graph classification aims to learn a domain whereas node classification aims to learn a function over this domain. Note that $\mathcal{H}(V)$ is isomorphic to $\mathbb{R}^{|V|}$, and thus can be endowed an Hilbert structure using the canonical inner product.

Similarly to the Euclidean case, classification admits a continuous counterpart called regression, and one can define **node regression** as the task

$$\Phi : \begin{cases} \mathcal{H}(V) & \to \mathbb{R} \\ z & \mapsto \Phi(z) \end{cases}$$

Typical question would be "*What is the energy $z(i)$ of the atom i in the activity of the protein G?*"

3 Invariance and Equivariance

In the context of graph learning, we are interested in the behavior of the function Φ with respect to the structure of the graph. In particular, we are interested in the behavior of Φ with respect to the permutation of the vertices.

3.1 Reminders on groups

A **group** is a couple $(\mathcal{G}, \circ)$ where $\mathcal{G}$ is a set and $\circ$ a binary operation on $\mathcal{G}$. This (internal) binary operation satisfies three properties

- *Associativity.* For any $g_1, g_2, g_3 \in \mathcal{G}$, one has

$$(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3).$$

- *Existence of a unit element.* There exists a unique element $e \in \mathcal{G}$ such that for any $g \in \mathcal{G}$, one has

$$g \circ e = e \circ g = g.$$

- *Existence of an inverse.* For any $g \in \mathcal{G}$, there exists a unique $g^{-1} \in \mathcal{G}$ such that

$$g \circ g^{-1} = g^{-1} \circ g = e.$$

Groups have a central place in the study of symmetry in algebra, and the notion of invariance emerges directly from it. Basic examples of groups includes

- The real line endowed with the addition $(\mathbb{R}, +)$;
- The set of rotations endowed with the composition $(\mathrm{SO}(n), \circ)$;
- The set of permutations endowed with the composition $(\mathfrak{S}_n, \circ)$.

3.2 Group action

Without diving in the subtilites of representation theory, we can recall that one of the most important tool of group theory is the notion of group action. A **(left) group action** $\cdot$ of $(\mathcal{G}, \circ)$ onto a set $\mathcal{X}$ is a map from $\mathcal{G} \times \mathcal{X}$ to $\mathcal{X}$ that maps $(g, x) \in \mathcal{G} \times \mathcal{X}$ to $g \cdot x \in \mathcal{X}$ such that it is

- *associative*: for all $g_1, g_2 \in \mathcal{G}$, $x \in \mathcal{X}$, one has $g_1 \cdot (g_2 \cdot x) = (g_1 \circ g_2) \cdot x$;
- *identity law*: for all $x \in \mathcal{X}$, one has $e \cdot x = x$.

A natural group action encountered in geometry and imaging is the (natural) action of the 2D (resp. 3D) rotations SO(2) (resp. SO(3)) onto the ambient space $\mathbb{R}^2$ (resp. $\mathbb{R}^3$). Intuitively, this is simply applying a rotation to a point in $\mathbb{R}^n$, and by extension to a subset $S \subseteq \mathbb{R}^2$.

Another well-known example is the action of $\mathbb{S}_n$ on $\mathbb{R}^n$ defined by

$$\sigma \cdot x = \begin{pmatrix} x_{\sigma(1)} \\ \vdots \\ x_{\sigma(n)} \end{pmatrix}$$

Remark that a group can act on different sets, for instance $\mathfrak{S}_n$ can act on the finite set $\{1, \ldots, n\}$ by $\sigma \cdot m = \sigma(m)$, and a group can act with two different actions on the same set (replace σ by σ^{-1}).

These two examples of actions have a common property: they are **linear** group actions. Linear group actions are probably the most fundamental examples of actions, and a whole field is dedicated to their studies: *group representations*. A group action is said to be linear if $\mathcal{X}$ is a vector space, for all $g \in \mathcal{G}$, $x, x' \in \mathcal{X}$ and for all $\alpha, \beta \in \mathbb{R}$,

$$g \cdot (\alpha x + \beta x') = \alpha(g \cdot x) + \beta(g \cdot y).$$

3.3 Invariance and Equivariance

Consider a function $f : \mathcal{X} \to \mathcal{Y}$. We say that f is **$\mathcal{G}$-invariant** if for any $g \in \mathcal{G}$, $x \in \mathcal{X}$, one has

$$f(g \cdot x) = f(x).$$

For instance, if $\mathcal{G} = \mathrm{SO}(2)$ is the group of 2D rotations and $\mathcal{X} = \mathbb{R}^2$ the Euclidean plane, then the Euclidean norm $f(x) = \|x\|$ is $\mathrm{SO}(2)$-invariant: given any rotation $r_\theta \in \mathrm{SO}(2)$, one has $\|r_\theta(x)\| = \|x\|$.

Consider now a group $\mathcal{G}$ acting on $\mathcal{X}$ and also[2] acting on $\mathcal{Y}$. We say that f is **$\mathcal{G}$-equivariant** if for any $g \in \mathcal{G}$, $x \in \mathcal{X}$, one has

$$f(g \cdot x) = g \cdot f(x).$$

Important examples includes function that index-based in $\mathbb{R}^n$. For instance, if $\mathcal{G} = \mathfrak{S}_n$ and $\mathcal{X} = \mathcal{Y} = \mathbb{R}^n$, then $f(x) = \arg\min_{1 \leq i \leq n} x_i$ is $\mathfrak{S}_n$-equivariant: $f(\sigma \cdot x) = \sigma(\arg\min_{1 \leq i \leq n} x_i)$.

4 A Fourier look at graph signal processing

At this point, we are quite far from concrete application in machine learning. A first question is: what are the invariance or equivariance that we want to enforce? In the following section, we will consider the important example of shift-invariance as a building block of Convolutional Neural Networks.

4.1 Convolution

The convolution is a fundamental operation in signal processing. Given two real functions g, h, let say Lebesgue-integrable, one defines the **convolution** $g * h$ as

$$(g * h)(t) = \int_{\mathbb{R}} g(t - \tau) h(\tau) d\tau.$$

One may look at g as a *signal* and h as a *filter* eventhough the two functions may play reversed-role. An important interpretation of the convolution is to see it as a local averaging method: it is indeed a sort of generalization of a moving average.

For our purpose, the important property of convolution is that it is *shift-equivariant*. This means that if we shift the input signal, the output signal

[2] In full generality, we may consider different groups but we will not need this generalization for our purpose of studying graph neural networks.

is shifted in the same way: if $v \in \mathbb{R}$, $g, h \in L^1(\mathbb{R})$, then,

$$(T_v \cdot g) * h = T_v \cdot (g * h).$$

where T_v is the translation operator defined by $T_v \cdot g(t) = g(t - v)$.

In traditional signal processing, such filters h are given: Gaussian blur, sharpening, etc. In contrast, in machine learning, and especially for CNNs, such filter are *learned* during the training procedure (see course Introduction to Deep Learning).

Our issues here are

- What is a convolution on a graph?
- Even more basic, what is shift-equivariance on a graph?

There is no universal answer to these two questions.

4.2 Convolution and Fourier

For an integrable function $g : \mathbb{R} \to \mathbb{R}$, its **Fourier transform** $\mathcal{F}[g] : \mathbb{R} \to \mathbb{C}$ is defined as

$$\mathcal{F}[g] : \omega \mapsto \int_{\mathbb{R}} g(t) e^{-2i\pi\omega t} dt.$$

On the $L^2(\mathbb{R})$ space, $\mathcal{F}[g]$ might be view as the inner product between g and the complex exponentials

$$\mathcal{F}[g](\omega) = \langle g, \exp_{-2i\pi\omega} \rangle_{L^2(\mathbb{R})}$$

The key property used in signal processing is that Fourier transform *diagonalize* the convolution: for any $g, h \in L^2(\mathbb{R})$, then

$$\mathcal{F}[g * h] = \mathcal{F}[g]\mathcal{F}[h].$$

I show in Figure **??** an example of the convolution of two functions and its Fourier transform.

width=.9]intro-to-gnn-convolution.svg

Figure 2: Convolution of an image by a low-pass filter.

The goal hence is to define what is a "Fourier transform" on graphs. To this end, we take yet another interpretation: the complex exponentials $\exp_{-2i\pi\omega} : t \mapsto e^{-2i\pi\omega t}$ are the eigenfunctions of the **Laplacian** operator on the torus $S^1 = \mathbb{R}/\mathbb{Z}$. If $f = \exp_{-2i\pi\omega}$, then $f'(t) = -2i\pi\omega f(t)$ and

$$f''(t) = (-2i\pi\omega)^2 f(t) = -4\pi^2\omega^2 f(t).$$

5 Graph Laplacian

Definition 1. Graph Laplacians *The **(combinatorial) graph Laplacian** of a graph G is defined as*

$$L = D - A$$

where A is the adjacency matrix of G, and $D = \mathrm{diag}(d_i)_{1\leq i\leq n}$ is the degree matrix defined by d_i being the degree of the vertex i.

*The **(normalized) graph Laplacian** of a graph G is defined as*

$$\mathcal{L} = D^{-1/2}LD^{-1/2} = \mathrm{Id} - D^{-1/2}AD^{-1/2}.$$

The combinatorial graph Laplacian is a semidefinite positive symmetric matrix associated to the quadratic form

$$\langle Lz, z\rangle = \sum_{i\sim j}(x_i - x_j)^2.$$

The spectral graph theory is more often concerned with the normalized version (??, a). It has several interesting properties (among many others):

Proposition 1. *Let G a graph and $\mathcal{L}$ its associated normalized Laplacian. Then,*

- *$\mathcal{L}$ is a differential operator on the space of function $g : V \to \mathbb{R}$:*

$$(\mathcal{L}g)(i) = \frac{1}{\sqrt{d_i}}\sum_{j\sim i}\left(\frac{g(i)}{\sqrt{d_i}} - \frac{g(j)}{\sqrt{d_j}}\right)$$

- *There exists a matrix $M \in \mathbb{R}^{n\times m}$ such that $\mathcal{L} = MM^\top$ and M is called the normalized incidence matrix that reads*

$$M_{i,e} = \begin{cases} \pm 1/\sqrt{d_i} & \text{if } i \in e \\ 0 & \text{otherwise.} \end{cases}$$

- *$\mathcal{L}$ is a real symmetric operator.*

An other important fact is that seeing $\mathcal{L}$ as a map of G, we can say that it is $\mathfrak{S}_n$-equivariant: if one reorders the nodes, it is sufficient to reorder lines and columns to get the new Laplacian matrix.

5.1 Graph Fourier Transform

The last point of Proposition prp-laplacian-graph, namely the fact that $\mathcal{L}$ is a real symmetric operator is fundamental. The spectral theorem tells us that there exists an orthonormal $U = [u_0 \cdots u_{n-1}] \in \mathbb{R}^{n \times n}$ and n real values $\lambda_0 \leq \cdots \leq \lambda_{n-1} \in \mathbb{R}$ such that

$$\mathcal{L} = UDU^T \quad \text{where} \quad D = \text{diag}(\lambda_0, \cdots, \lambda_{n-1}) \in \mathbb{R}^{n \times n}.$$

Observe that U is not necessarly uniquely defined, and that $\lambda_0 = 0$ associated to the eigenvector 1_n.

Recall that the Fourier transform on $L^2(\mathbb{R})$ is defined as

$$\mathcal{F}[g](\omega) = \langle g, \exp_{-2i\pi\omega} \rangle_{L^2(\mathbb{R})},$$

where $\exp_{-2i\pi\omega}$ are the eigenvectors of the Laplacian. Analogously, we define the Fourier transform on a graph (GFT) as follow:

Definition 2. *Let G a graph, $\mathcal{L}$ its normalized Laplacian and $\mathcal{L} = UDU^T$ its eigendecomposition. Let $z : V \to \mathbb{R}$, i.e, $z \in \mathbb{R}^n$ a signal on G. Then, the* ***graph Fourier transform*** *of z is defined as*

$$\mathcal{F}[z](\lambda_k) = \langle z, u_k^\top \rangle_{\mathbb{R}^n} = \sum_{i=1}^{n} z(i) u_k^\top(i) \quad \textit{for} \quad k = 1, \ldots, n-1.$$

We can rewrite it in matrix form as $\mathcal{F}[z] = U^\top z$. An important feature of the GFT is that there exists an inverse GFT, defined similarly to the real case:

$$z(i) = \sum_{k=0}^{n-1} \mathcal{F}[z](\lambda_k) u_k(i) \quad \text{for} \quad i = 1, \ldots, n.$$

5.2 Filtering on graphs

Given the definition of the GFT, how to filter a signal? The idea is to act on the eigenvalues of the eigendecomposition, and let eigenvectors untouched. So given a filter $h : \mathbb{R} \to \mathbb{R}$, the filtering of $z \in \mathbb{R}^n$ by h is given by:

$$h * z = \mathcal{F}^{-1}[h(D)\mathcal{F}[z]] = Uh(D)U^\top z,$$

where $h(D)$ is applied pointwise. An important point: despite the notation, the filter $h * z$ is *not* a regular convolution. Observe that $z \mapsto h * z$ is $\mathfrak{S}_n$-equivariant.

In practice, diagonalizing the Laplacian is too costly. What people use is *polynomial, or analytical, filters* of the Laplacian. Given a polynomial, or formal serie, $h = \sum_{k\geq 0} \beta_k X^k \in \mathbb{R}[[X]]$, we let (possibly not-well defined in the case of formal serie)

$$h * z = \sum_{k\geq 0} \beta_k L^k z.$$

Note that from a theoretical perspect, it is still only application on the eigenvalues. Figure fig-conv-graph illustrates the filtering of a graph signal by a low-pass filter.

width=.9]intro-to-gnn-conv-graph.svg

Figure 3: Convolution of a graph by a low-pass filter.

6 Graph Neural Network

6.1 (Spectral) Graph Convolutional Network

Up to some subtles or notational differences, you probably encounter CNNs as parametric function defined layer-wise by a combination of

- non-linearities such as ReLU;
- convolution layers mapping a tensor to another;
- pooling (or subsampling) reducing the dimension using for instance a max- or mean-pooling;
- fully connected layers (Multi-Layer Perceptron, MLP) to produce an output.

In the context of GNNs, non-linearities will be the same (typically Lipschitz function different from the identity), convolution will be used as defined in the previous section, MLPs will be the same. But what about pooling? This is (mostly) an open question in the GNNs on how to produce good and efficient pooling operation, also known as graph coarsening. These architectures were defined around 2015, by (??, a) and (??, a).

A GCN with $M + 1$ layers is defined as follows:

1. The signal at the input layer is some *node features* $Z^{(0)} = Z$ with dimension $d_0 = d_z$ and built up columns $z_j^{(0)} \in \mathbb{R}^n$.

2. For all $\ell = 1, \dots, L$, the signal $Z^{(\ell+1)} \in \mathbb{R}^{n \times d_{\ell+1}}$ at layer $\ell + 1$ is obtained by propagation of the signal $Z^{(\ell)} \in \mathbb{R}^{n \times d_\ell}$ at layer ℓ with respect to analytic filters of the normalized Laplacian:

$$z_j^{(\ell+1)} = \rho\left(\sum_{i=1}^{d_\ell} h_{ij}^{(\ell)}(\mathcal{L}) z_i^{(\ell)} + b_j^{(\ell)} 1_n\right) \in \mathbb{R}^n, \quad \forall j = 1 \dots d_{\ell+1}. \tag{1}$$

3. The output depends on the notion of invariance that one wishes to preserve.

   - $\mathfrak{S}_n$-*equivariance* (node classification). In this case, the output of the GCN is a node-signal $\Phi_G(Z) \in \mathbb{R}^{n \times d_{\text{out}}}$ defined as

     $$\Phi_G(Z) = \text{MLP}_\theta(Z^{(L)}),$$

     where MLP_θ is a MLP **applied row-wise** to the signal $Z^{(L)}$.
   - $\mathfrak{S}_n$-*invariance* (graph classification). In this case, the output is a single vector $\bar{\Phi}_G(Z) \in \mathbb{R}^{d_{\text{out}}}$ given as a *global pooling*

     $$\bar{\Phi}_G(Z) = \frac{1}{n} \sum_{i=1}^{n} \Phi_G(Z)_i.$$

6.2 Invariance and Equivariance of GCNs

In the definition of GCNs, it was roughly said that we can made the output either $\mathfrak{S}_n$-equivariant or $\mathfrak{S}_n$-invariant. But what do we mean exactly by that? We said that a permutation σ acts onto the space of GCNs with the action $\sigma \cdot \Phi_G$ defined as

$$\forall Z \in \mathbb{R}^{n \times d_0}, \quad (\sigma \cdot \Phi_G)(Z) = \Phi_{\sigma \cdot G}(\sigma \cdot Z),$$

where we defined the action on the graph and the signal as

$$\sigma \cdot G = (\sigma \cdot V, \sigma \cdot E)$$
$$\sigma \cdot Z = \begin{pmatrix} (z_1)_{\sigma(1)} & \cdots & (z_{d_0})_{\sigma(1)} \\ \vdots & & \vdots \\ (z_1)_{\sigma(n)} & \cdots & (z_{d_0})_{\sigma(n)} \end{pmatrix}$$

The action on the group is more easily defined in fact on the adjacency matrix. For any $\sigma \in \mathfrak{S}_n$, let P_σ be the associated permutation matrix. Then, one can defined

$$\sigma \cdot A = P_\sigma A P_{\sigma^{-1}} = P_\sigma A P_\sigma^\top.$$

Cutting the algebraic details, these two definitions are perfectly equivalent.

We can now state the result

Proposition 2. *For any G, and filters h, Φ_G is $\mathfrak{S}_n$-equivariant and $\bar{\Phi}_G$ is $\mathfrak{S}_n$-invariant: for all $\sigma \in \mathfrak{S}_n$, one has*

$$(\sigma \cdot \Phi_G)(Z) = \sigma \cdot \Phi_G(Z),$$
$$(\sigma \cdot \bar{\Phi}_G)(Z) = \bar{\Phi}_G(Z).$$

Proof. See for instance (??, , Proposition 1). The proof of $\mathfrak{S}_n$-equivariant is based on three facts:

1. The map $\mathcal{L} : A \mapsto \mathrm{Id} - D^{-1/2}AD^{-1/2}$ is $\mathfrak{S}_n$-equivariant;
2. Hence the map $h \circ \mathcal{L} : A \mapsto h(\mathcal{L}(A))$ is $\mathfrak{S}_n$-equivariant;
3. Permutations matrices commutes pointwise with entrywise activation function.

To get the $\mathfrak{S}_n$-invariance, one has to see that agregaging with a sum is a universal way to construct a $\mathfrak{S}_n$-invariant function from $\mathfrak{S}_n$-equivariant function. □

7 Another paradigm: Message-Passing Graph Neural Networks

Before ending this nose, let me mention that around 2017, (??, a) and (??, a) proposed a new architecture, or at least a new interpretation, that "forget" the spectral interpretation of GCNs. It is closer in the spirit to original graph neural network model proposed by (??, a), and is now the standard way to think of GNNs. I still believe that the spectral interpretation highlights important properties of GNNs, and is a good way to understand the behavior of GNNs.

Instead of a spectral interpretation, a neighborhood aggregation, or so-called **message-passing** architecture is proposed:

$$z_j^{(\ell+1)} = \gamma^{(\ell)} \left(z_j^{(\ell)}, \square_{i \in \mathcal{N}(j)} \rho^{(\ell)} \left(z_j^{(\ell)}, z_i^{(\ell)}, e_{j,i} \right) \right),$$

where

- $z_j^{(\ell)}$ is the node feature at layer ℓ of node j;

- $\gamma^{(\ell)}$ (the **update** function) and $\rho^{(\ell)}$ (the **message-passing** function) are learnable functions such as traditional MLPs;
- $\square$ is an **agregation** function such as sum $\sum$ or maximum max;
- $e_{j,i}$ is a (potential) edge features.

In many papers, this is presented in the equivalent form (often with multisets for the aggregation)

$$\begin{aligned} m_j^{(\ell)} &= \mathrm{MSG}^{(\ell)}\left(\{z_i^{(\ell)} : i \in \mathcal{N}(j) \cup \{j\}\}\right) \\ a_j^{(\ell)} &= \mathrm{AGG}^{(\ell)}\left(\{m_i^{(\ell)} : i \in \mathcal{N}(j)\}\right) \\ z_j^{(\ell+1)} &= \mathrm{UPD}^{(\ell)}\left(z_j^{(\ell)}, a_j^{(\ell)}\right). \end{aligned}$$

Let me mention several important examples of MPGNNs:

- *Convolutional GNN.* Wait what, didn't we called them spectral GNNs? Yes, but we can look at them in a slightly more general way

$$z_j^{(\ell+1)} = \gamma^{(\ell)}\left(z_j^{(\ell)}, \square_{i\in\mathcal{N}(j)} c_{i,j}\rho^{(\ell)}\left(z_i^{(\ell)}\right)\right),$$

  In practice, the community consider a more constrained version:

$$z_j^{(\ell+1)} = \mathrm{ReLU}\left(\frac{1}{|\mathcal{N}(j)|}\sum_{i\in\mathcal{N}(j)} W^{(\ell)} z_i^{(\ell)}\right),$$

  where $W^{(\ell)}$ is a learnable matrix. This can be even more precisely specified (in matrix form) as

$$Z^{\ell+1} = \mathrm{ReLU}(D^{-1}AZ^{\ell}W^{\ell,\top}),$$

  or in normalized form as

$$Z^{\ell+1} = \mathrm{ReLU}(D^{-1/2}AD^{-1/2}Z^{\ell}W^{\ell,\top}),$$

- *Attentional GNN.*

$$z_j^{(\ell+1)} = \gamma^{(\ell)}\left(z_j^{(\ell)}, \square_{i\in\mathcal{N}(j)} a\left(z_j^{(\ell)}, z_i^{(\ell)}\right)\right),$$

  where a is a learnable *self-attention* function.

Without diving into the details, I want to attract your attention that when dealing with a graph with only self-loop, or when $E = \mathcal{P}_2(V)$, we obtain two popular architectures in the litteratures.

- When $\mathcal{N}(i) = \{i\}$ – so not a classical graph, you have to add a self-loop – the convolutional architecture boils down to the *Deep Sets* architecture (??, a).
- On the other extreme, when $i \sim j$ for all i, j (again with self-loop), the attentional GNN is linked to the *Transformer* (??, a) architecture.