UNIVERSITY OF
CAMBRIDGE
Department of Computer
Science and Technology

Sheaf-based Positional Encodings for Graph Neural Networks

Yu He

Murray Edwards College

May 2023

Submitted in partial fulfillment of the requirements for the
Computer Science Tripos, Part III

Total page count: 50

Main chapters (excluding front-matter, references and appendix): 40 pages (pp 1–40)

Main chapters word count: 11670

Methodology used to generate that word count:

```
$ texcount -inc -sum=1,1,1,0,0,0,0 main.tex > wordcount.tex
```

(1) "%TC:group table 0 1" and "%TC:group tabular 1 1" were used to include tables.

(2) "%TC:ignore" and "%TC:endignore" were used to exclude abstract, acknowledgements, bibliography, and appendices.

Declaration

I, Yu He of Murray Edwards College, being a candidate for the Computer Science Tripos, Part III, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: *Yu He*

Date: *26/05/2023*

Abstract

Graph Neural Networks (GNNs) apply deep learning techniques to graph-structured data, leveraging the relational information between entities. However, the expressive power of GNNs is limited by the local interactions between connected nodes, causing GNNs to produce indistinguishable node embeddings when two nodes share similar local neighbourhoods. Moreover, GNNs can fail to distinguish structurally distinct graphs when a set of identical node embeddings are aggregated.

To address this issue, we use *positional encodings* to inform the nodes of their global positions in the graph, breaking the locality constraints. Positional encodings can provide global structural information to help the nodes differentiate from each other, and hence improve the expressive power of GNNs in identifying substructures and distinguishing non-isomorphic graphs. Furthermore, positional encodings are crucial for the effectiveness of Graph Transformers. Graph Transformers apply an attention mechanism over a fully-connected graph, losing the input graph topology. Therefore, positional encodings are necessary to complement Graph Transformers with the structural information from the original graph.

In this project, we propose a novel form of positional encodings based on a structure from geometry and topology, namely a *sheaf*. The sheaf Laplacian generalises the graph Laplacian, which is the basis for many existing positional encoding designs. The sheaf Laplacian can encode the algebraic and topological structure of the graph, producing a more expressive positional embedding space. We investigate the use of pre-computed and learnt sheaf-based positional encodings, analysing their trade-offs between computational efficiency and expressivity.

We demonstrate that sheaf-based positional encodings outperform current graph Laplacian positional encodings on node-level and graph-level tasks, indicating their local and global effectiveness. We also show that our positional encoding design is model-agnostic and test it across architectures, including Graph Transformers. Our work contributes to the study of sheaves within the graph learning context, motivating further investigation into designing novel GNN techniques using concepts from geometry and topology.

Acknowledgements

This project would not have been possible without the unwavering support of my supervisors, **Dr Cristian Bodnar** and **Prof. Pietro Liò**. Cris has been an amazing supervisor, generously offering me support both within and beyond the scope of the project. I am truly grateful for his patience and expertise in guiding my project and his understanding when my schedule turned overwhelming. Moreover, I am honoured to build on his works on the intersection of topology and GNNs, which continues to inspire me to explore this fascinating area in my future research. For Pietro, I owe an immense dept of gratitude to him. Pietro has not only provided me with so much support academically since Part II, but also has been offering extensive help and care to make sure I grow into my best. He is such a remarkable academic role model that I deeply admire. I can never pay back as much as what he has offered to me.

I would also like to take this opportunity to thank the many wonderful people whom I met at CL outside of this project, without whom I would never have become the person I am today.

First of all, I would like to express my heartfelt gratitude to **Dr Petar Veličković**, who was the person that led me onto the path of geometric deep learning. Petar accepted my request when I was just a Part IB student anxiously reaching out for a project supervisor. It was during lockdown when I had to live on the hope for this future research project to pass the exam period. Since then, everything has gone upwards for me. I finally got the chance to do something that I genuinely enjoy. I am forever thankful for Petar's kindness and trust in me. Needless to say, his passion for research and professionalism in delivering those fantastic talks and lectures will always be my inspiration. I hope that one day I can pass down his positive impact on me to motivate more people to pursue their dreams.

Next, I would like to thank **Andreea Deac**, who remains such a special person in my life. Andreea was my Part II project supervisor, but I had known her name long before that. It was her post on the college website that made me choose Medwards. Later in my first year, I heard about her successful research from our previous DoS. I then emailed the DoS to seek advice on how to embark on a research career like hers. I could never have imagined that she would come to supervise me two years later. Andreea has been supporting me with extraordinary dedication since day one, from my project to

internship, publication, and PhD applications. It was through that year working with her that I rebuilt my confidence in myself and continued to hold on to what I believed. I have talked about this many times, but I could never express my gratitude to her in exact words with the same amount of sincerity as I have in my heart.

Furthermore, I am also very grateful to **Sophie Xhonneux**, who was from CL and supervised me for my summer internship at Mila, during which I had a wonderful time. Sophie also kindly offered much earnest advice and support for my PhD applications, for which I am truly thankful. Additionally, I would also like to thank **Dobrik Georgiev** and **Chaitanya Joshi** for warmly offering to help me with my research and applications. Moreover, I would like to extend my special thanks to **Dr Challenger Mishra**, who provided me with encouragement and autonomy to explore the intersection of physics and geometry in machine learning.

Last but not least, I would like to thank **Han Xuanyuan**. Although CL is relatively small in size, it took us three years to formally introduce ourselves to each other at a guest talk invited by Pietro. From the Tripos days to the "post-mortem," he has been a beacon of light in my life, embracing me with his tenderness. He is someone of exceptional kindness, determination, and brilliance. I am deeply indebted to have someone like him who unconditionally accepts me, respects me, and supports me.

Contents

Chapter 1

Introduction

Deep learning is a flourishing field in which neural networks are used to learn patterns from high-dimensional data. Many real-world datasets have inherent graph structures, where data entities are connected through their intrinsic relationships. However, a feed-forward neural network does not utilise such relational information in its predictions, inhibiting its ability to generate a precise hypothesis space. This limitation motivates the introduction of Graph Neural Networks (GNNs) [1, 2, 3], which directly operate on graphs to preserve the topology. GNNs have demonstrated their effectiveness across a wide range of applications, including protein interface prediction [4], social networks [5], and knowledge graphs [6].

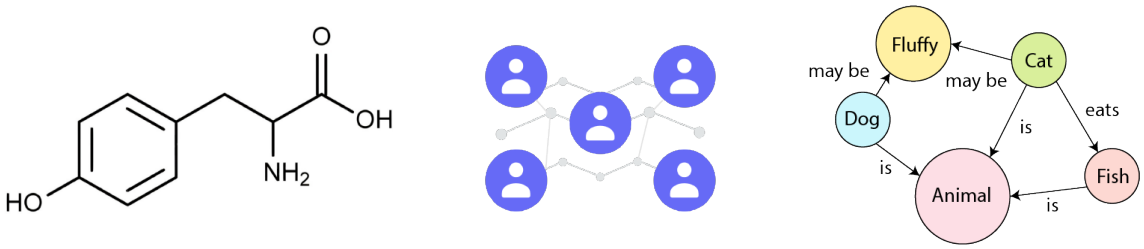
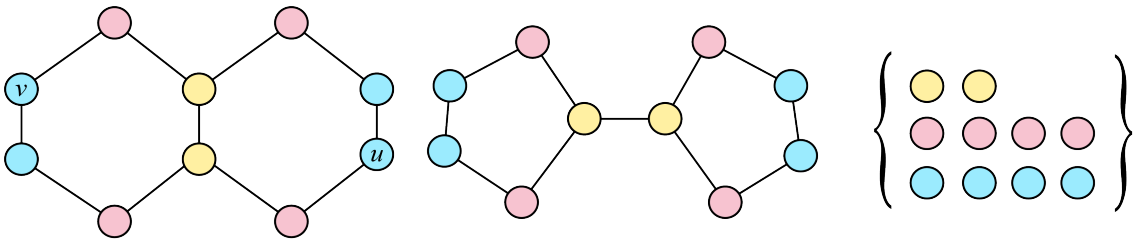


Figure 1.1: Examples of real-world data with graph structures.

However, GNNs suffer from limited expressive power [7]. In general, GNNs follow a message-passing paradigm [2, 8], where a node gathers messages from its neighbours. The locality is essential for scalability, but it also constrains the knowledge of each node to its local structure. This limited view poses a challenge for GNNs to differentiate distinct nodes. For instance, in Figure 1.2(a), even though two nodes v and u are far apart in the graph, GNNs may produce the same node embedding as they share an identical local structure. While node features can help to distinguish the nodes, they are not guaranteed to be unique. This issue can be particularly problematic when projected onto real-life applications, such as molecules and social media, where it is common to have nodes that share similar local neighbourhoods.

In addition, GNNs can fail to distinguish some graphs that are structually different. Some



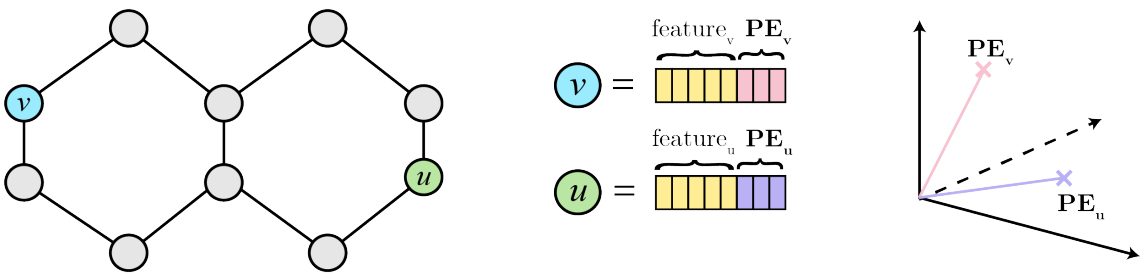
(a) decalin (b) bicyclopentyl (c) graph embedding

Figure 1.2: (a) and (b) are a pair of structurally distinct graphs, representing two chemical compounds [9]. Same colour indicates same node embedding generated by a GNN. (c) A GNN produces an identical graph embedding for the two graphs.

tasks require GNNs to produce a graph embedding, which can be obtained by aggregating all node embeddings through a graph-level pooling layer. Ideally, graph embeddings should only be identical if the graphs are structurally the same. However, as exemplified in Figure 1.2, two structually different graphs may share the same set of node embeddings due to the locality constraint of GNNs, receiving an identical graph embedding. This problem can lead to false graph-level predictions, because structurally different graphs tend to have distinct labels.

1.1 Motivation

To improve the expressive power of GNNs, we focus on using *positional encodings (PEs)* to make the nodes aware of their global positions in the graph, as illustrated in Figure 1.3. PEs have been used in natural language processing to encode the token positions in a sequence. Similarly, in the context of GNNs, we construct PEs to encode the node positions in a graph. However, nodes in a graph lack a canonical order to assign sequential indexing, making the definition of PEs non-trivial.



(a) (b) (c)

Figure 1.3: (a) Two nodes v and u share the same local neighbourhood and node feature. (b) We can augment them with positional encodings (PEs) to make them distinguishable. (c) Our goal is to design a positional embedding space to encode the graph structure.

Furthermore, PEs play an essential role in Graph Transformers, which are variants of

GNNs that generalise the Transformer [10] architecture onto graphs. Just as how Transformers treat each token in the sequence as independent and augment the token with its positional information using PEs, Graph Transformers apply the attention mechanism over a fully-connected graph, requiring PEs to make up the loss of the graph topology. Hence, the design of PEs is crucial to enhance the expressivity of Graph Transformers.

The eigendecomposition of the *graph Laplacian* provides an embedding space that respects the graph topology [11], making them a popular form of PEs [12, 13]. However, a problem arises when there exist complex relationships between the nodes. For example, unlike homophilic graphs, where nodes with similar representations are positioned closer ('like attracts like'), heterophilic graphs have dissimilar nodes connected. The graph Laplacian only encodes the adjacency information and is independent of the node data. Consequently, the positional embedding space constructed from the graph Laplacian may become unsuitable for these graphs.

We propose using the *sheaf Laplacian* to construct more expressive PEs that address this issue. A cellular sheaf is a mathematical structure from geometry and topology. It has the ability to encode both the algebraic and topological structures parameterised by the node data, allowing more complex relationships between the nodes to be expressed. Such a property gives the sheaf Laplacian a potential to construct a positional embedding space that incorporates both structural and semantic information of the graph, adapting it to more challenging domains such as heterophilic graphs.

1.2 Contributions

Our contributions can be summarised as follows:

- We propose a novel design of positional encodings using sheaf theory. The sheaf-based PEs outperform current methods based on the graph Laplacian, demonstrating their effectiveness on both node-level and graph-level tasks.
- We design two forms of sheaf-based PEs. The first method computes the sheaf Laplacian at preprocessing time. We also augment it with an evolvable mechanism to address its limitation on less informative input data.
- We propose a second design by learning a sheaf from the input data and address the challenges in training-time eigendecomposition. We also analyse different sheaf types and the effect of the Laplacian normalisation.
- Empirically, we evaluate our model-agnostic sheaf-based PEs across architectures, covering both sparse GNNs and fully-connected Graph Transformers. We also compare PEs on a set of node-level tasks, in addition to the graph-level tasks, which have been the focus of previous works.

Chapter 2

Background & Related Work

This chapter explains why PEs are useful for GNNs by providing the theoretical foundation to understand this project and discussing related works. We begin by introducing the theory of GNNs and the message-passing mechanism employed by them (Section 2.1). We then explain how this mechanism limits the expressivity of GNNs (Section 2.2). We also discuss several solutions to address this issue, motivating the use of PEs. Moreover, we emphasise the importance of PEs for Graph Transformers (Section 2.3). This is followed by a review of current PE methods (Section 2.4), establishing the connection between our sheaf-based PEs and related works.

2.1 Graph Neural Networks

Graph Neural Networks (GNNs) [1, 2, 3] are neural networks that operate directly on graph-structured data, leveraging the relational information between the nodes. An input graph can be represented as $G = (V, E)$, where V is the set of nodes and E is the set of edges. For each node $v \in V$, its node representation is a vector $\mathbf{h}_v \in \mathbb{R}^{d_h}$, where d_h is the dimension. Optionally, we can have edge representations to denote the relations between nodes $\mathbf{e}_{v \to u} \in \mathbb{R}^{d_e}$, with d_e as the dimension.

The inference tasks for GNNs can be classified into three main categories, depending on the level where the prediction is made:

- **Node-level**: These tasks predict a label for each node in the graph. The goal is to learn a proper representation $\mathbf{h}_v$ for each node $v \in V$. One example is the classification of documents in a citation network.
- **Edge-level**: Edge-level tasks predict the property or existence of edges. This requires learning a proper representation $\mathbf{e}_{v \to u}$ for each edge $e_{v \to u} \in E$. Examples include determining whether two users in a social network are friends.
- **Graph-level**: Some tasks predict the label of the whole graph. This involves

mapping the graph to a graph embedding $\mathbf{g}_G$. For instance, the prediction of a molecule's chemical property.

2.1.1 Message-passing paradigm

GNNs generally follow a message-passing paradigm [2, 8] as illustrated in Figure 2.1, where nodes exchange messages with their connected neighbours to update their own representations. If we omit edge features for clarity, each node $v \in V$ updates its node representation $\mathbf{h}_v$ via:

$$\mathbf{h}_v^{l+1} = \phi^l\Big(\mathbf{h}_v^l, \bigoplus_{u \in \mathcal{N}(v)} \psi^l(\mathbf{h}_v^l, \mathbf{h}_u^l)\Big) \tag{2.1}$$

where l is the layer number, $\mathcal{N}(v) = \{u \in V | (u, v) \in E\}$ is the one-hop neighbourhood of node v, and ϕ^l, ψ^l are commonly implemented as Multilayer Perceptrons (MLPs).

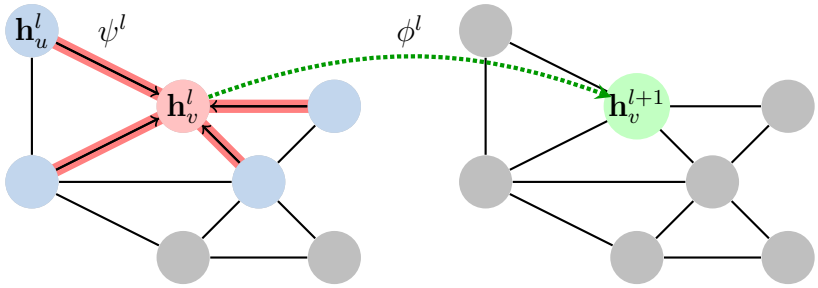


Figure 2.1: In one message-passing step, a node gathers messages from its neighbours (left) and updates its node feature using the aggregated message (right).

To interpret Equation 2.1, at each layer l, a node computes a message from its connected neighbours using a message function ψ^l, then aggregates the message with a permutation-invariant aggregation operator $\bigoplus$. Finally, the node uses the message to update its node representation with a readout function ϕ^l.

The permutation invariance property of the aggregation operator $\bigoplus$ is essential for the local operation on graphs. In practice, node representations are stacked together as a matrix $\mathbf{H} \in \mathbb{R}^{|\mathcal{N}(v)| \times d_h}$ as input to $\bigoplus$. Since there is no canonical order for the neighbouring nodes, the aggregated results should be identical regardless of the node order, i.e. $\bigoplus(\mathbf{H}) = \bigoplus(\mathbf{PH})$ for any permutation matrix $\mathbf{P}$. Therefore, $\bigoplus$ should be permutation invariant, such as element-wise sum, mean, or max. Below, we briefly describe the three GNN models used in the evaluation.

2.1.2 Graph Convolutional Network (GCN)

GCN [1] is one of the most popular GNN frameworks due to its efficiency. Its convolution operation is based on a spectral filter for node signals in the Fourier domain, which is

then localised to the 1-hop neighbourhood for better scalability. Formally, for each node $v \in V$, a GCN layer updates the node representation $\mathbf{h}_v$ via:

$$\mathbf{h}_v^{l+1} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \sqrt{\frac{1}{(\deg(v)+1)(\deg(u)+1)}} \mathbf{W}^l \mathbf{h}_v^l\right) \tag{2.2}$$

where l is the layer number, $\mathcal{N}(v)$ is the 1-hop neighbourhood of node v, $\deg(\cdot)$ denotes the node degree, $\mathbf{W}$ is a layer-wise weight matrix, and σ is a non-linear activation function.

2.1.3 Residual Gated Graph ConvNets (GatedGCN)

Inspired by the success of LSTM [14] and ConvNets [15], [16] proposes a natural extension onto graphs. They propose to use an edge gating mechanism to learn the importance of edges when passing messages. In GatedGCN, η_{vu} is a gate for an edge $u \to v$ defined as:

$$\eta_{vu} = \sigma(\mathbf{A}^l \mathbf{h}_v^l + \mathbf{B}^l \mathbf{h}_u^l) \tag{2.3}$$

where $\mathbf{A}$ and $\mathbf{B}$ are layer-wise weights, and σ is a non-linear activation function. The overall update rule additionally uses a residual mechanism:

$$\mathbf{h}_v^{l+1} = \mathrm{ReLU}\Big(\mathbf{U}^l \mathbf{h}_v^l + \sum_{(u,v) \in E} \eta_{vu} \odot \mathbf{V}^l \mathbf{h}_u^l\Big) + \mathbf{h}_v^l \tag{2.4}$$

where l is the layer number, and $\mathbf{U}, \mathbf{V}$ are layer-wise trainable weights.

2.1.4 Principal Neighbourhood Aggregation (PNA)

Most GNN architectures use a single aggregation operator ($\bigoplus$ in Equation 2.1), usually choosing from sum, mean, or max. PNA [17] was proposed to combine multiple aggregation schemes with degree-scalars for more expressive and robust GNNs. The overall aggregation function is defined as:

$$\bigoplus = \begin{bmatrix} \text{identity} \\ \text{amplification} \\ \text{attenuation} \end{bmatrix} \otimes \begin{bmatrix} \text{mean} \\ \text{max} \\ \text{min} \\ \text{std} \end{bmatrix} = \begin{bmatrix} S(V, \alpha = 0) \\ S(V, \alpha = 1) \\ S(V, \alpha = -1) \end{bmatrix} \otimes \begin{bmatrix} \text{mean} \\ \text{max} \\ \text{min} \\ \text{std} \end{bmatrix} \tag{2.5}$$

where $S(v, \alpha)$ is the degree-scalar for each aggregator controlled by α, and δ is a normalisation parameter computed over the training set:

$$S(v, \alpha) = \left(\frac{\log(\deg(v)+1)}{\delta}\right)^{\alpha}, \quad -1 \le \alpha \le 1 \tag{2.6}$$

$$\delta = \frac{1}{|\text{train}|} \sum_{G \in \text{train}} \sum_{v \in V} \log(\deg(v)+1) \tag{2.7}$$

2.2 Expressivity of GNNs

The message-passing paradigm is centred on pairwise communication within a local neighbourhood, providing high scalability. However, this locality also leads to a problem. GNNs can fail to distinguish structurally distinct graphs or even recognise simple substructures. This limitation of GNNs is well understood through the analogy to the Weisfeiler-Lehman test on graph isomorphism [7].

2.2.1 Weisfeiler-Lehman test

Graph isomorphism is an important concept in graph theory, capturing whether two graphs are topologically identical. It remains a challenging problem where no polynomial time solution exists [7].

Definition 1 (Graph isomorphism) *Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic ($G_1 \simeq G_2$) if there exists a bijection between their verticies $f : V_1 \to V_2$, such that f preserves the adjacency of verticies, i.e. if $(v, u) \in E_1$ then $(f(v), f(u)) \in E_2$.*

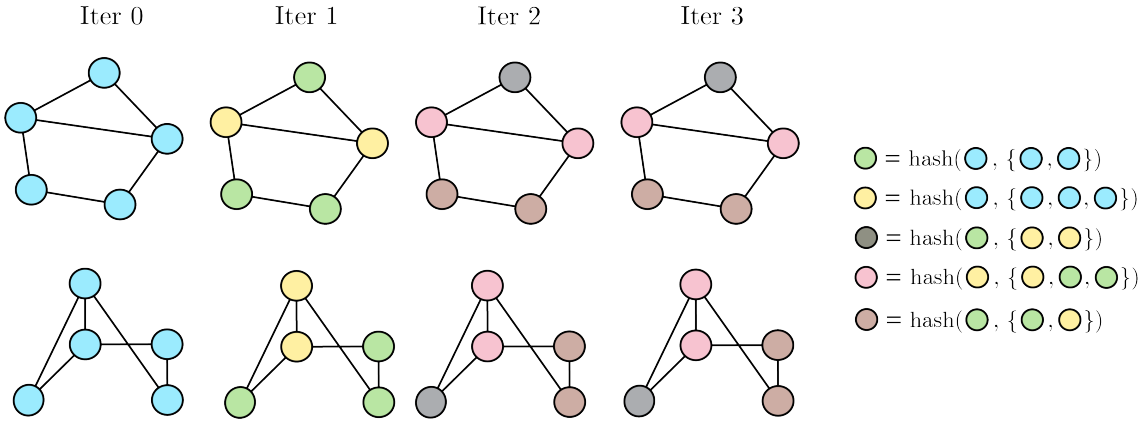


Figure 2.2: A pair of isomorphic graphs successfully identified by the 1-WL test. Example adapted from [18].

The Weisfeiler-Lehman test [19] is a polynomial-time algorithm to determine graph isomorphism for a certain class of graphs [20]. An example of running the 1-dimensional Weisfeiler-Lehman (1-WL) test is illustrated in Figure 2.2. At initialisation, each node $v \in V$ is assigned the same colour C_v^0. In each iteration, a node aggregates a multiset of colours from its neighbouring nodes, and generates a random hash as its new colour:

$$C_v^{t+1} = \text{hash}(C_v^t,\ \{C_u^t,\ \text{for } u \in \mathcal{N}(v)\}) \tag{2.8}$$

The algorithm converges when two graphs have different multisets of node colours, distinguishing them as non-isomorphic. Otherwise, the algorithm stops after n iterations, where n is the number of nodes.

2.2.2 GNNs are at most as powerful as the 1-WL test

We can observe the similarity between the update function of 1-WL test (Equation 2.8) and message-passing (Equation 2.1). They both aggregate information from a node's local structure and use it to update the node representation. In fact, GNNs are at most as powerful as the 1-WL test, as proven in [7, 21]:

Lemma 1 *Let G_1 and G_2 be any two non-isomorphic graphs. If a GNN $f : G \to \mathbb{R}^d$ produces different graph embeddings for G_1 and G_2, then the 1-WL test also decides that G_1 and G_2 are non-isomorphic.*

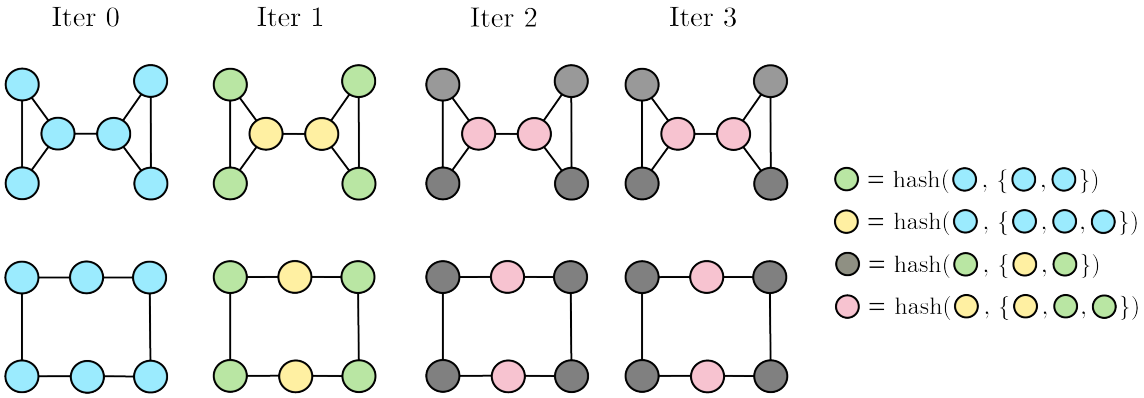


Figure 2.3: A pair of non-isomorphic graphs failed to be distinguished by the 1-WL test and a message-passing GNN.

Failed cases in distinguishing non-isomorphic graphs However, the 1-WL test is only a heuristic for the graph isomorphism test, failing to distinguish some non-isomorphic graphs, such as regular graphs [22, 23, 24]. One failed case is illustrated in Figure 2.3. For these graphs, the 1-WL test produces the same multiset of node colours. Similarly, GNNs generate an identical set of node representations, falsely identifying the graphs as isomorphic, leading to an identical graph-level prediction. This becomes an issue for real-world data such as molecules, where two non-isomorphic graphs commonly have different graph-level labels.

2.2.3 Increasing the expressivity of GNNs

Ideally, a maximally expressive GNN should only map the graphs to the same embedding if they are isomorphic, and produce different embeddings for non-isomorphic ones. Many efforts have been invested in increasing the expressive power of GNNs to surpass the 1-WL test. We motivate the use of positional encodings by comparing them with several existing approaches.

Higher-order iteractions The message-passing paradigm exchanges information in a pairwise fashion. Therefore, many solutions utilise higher-order interactions between some form of substructures or groups of nodes. One extension from the pairwise interaction is

the communication between k-tuples of nodes, which is analogous to the k-dimensional WL test [21]. On the other hand, hypergraphs, where a set of nodes are connected to each hyperedge, can also propagate higher-order signals [25, 26, 27]. Furthermore, node representations can be computed over simplicial complexes [28] and cellular complexes [29], lifting the graph into higher-order topological structures to facilitate hierarchical message-passing. However, the problem with these approaches lies in the high computational cost, making them less scalable to larger graphs.

Learning over subgraphs Another line of work learns over subgraphs, by redefining the neighbourhood through shared nodes in multiple subgraphs [30, 31]. The motivation comes from the observation that two hard-to-distinguish graphs often contain subgraphs that are easier to distinguish by GNNs. Similarly, the subgraph selection policy is non-trivial and very costly, which may require stochastic sampling to mitigate [31].

Positional encodings (PEs) Since message-passing limits a node's view to its local neighbourhood, we can use PEs [32, 12, 33, 34] to explicitly make it aware of its global position in the graph. PEs are usually concatenated to the node or edge features as inputs to the GNNs. The global knowledge from PEs can help GNNs to distinguish graphs and substructures.

This project focuses on using PEs to increase the expressive power of GNNs. PEs can be computed during preprocessing, and are usually more computationally efficient than other approaches. Furthermore, PEs only use the knowledge of the input graph, making them model-agnostic. The above benefits of PEs make them excellent candidates to address the expressivity limitation of GNNs.

2.3 Positional encodings for Graph Transformers

In addition to improving the expressive power of GNNs, PEs also play an essential role in Graph Transformers. Graph Transformers apply an attention mechanism over a fully-connected graph, losing the graph topology. Therefore, PEs are required to augment the nodes with their positional information in the input graph.

2.3.1 Graph Transformers

GNNs based on the message-passing mechanism are efficient and scalable by performing local computation on sparse graphs. However, the sparsity also puts GNNs at risk of some fundamental limitations. In sparse graphs, information may take many layers to propagate between nodes separated by a long distance. One problem with stacking multiple layers is over-smoothing [35], where node representations become indistinguishable through repeated aggregation. Another problem is over-squashing [36], where information

from an exponential number of nodes passes through a single node, leading to potential information loss as information gets compressed into a fixed-size vector.

Graph Transformers (GTs) [13, 34, 37, 38, 39, 40] were proposed to address these issues by generalising the Transformer architecture [10] to graphs. Take a graph G and its node representation matrix $\mathbf{H} \in \mathbb{R}^{n \times d_h}$, a Graph Transformer alternatively performs multi-head attention (MultiHead) and fully-connected feed-forward network (FFN) at each layer l to update the node representations:

$$\mathbf{H}^{l+1} := \text{FFN}(\text{MultiHead}(\mathbf{H}^l) + \mathbf{H}^l) \tag{2.9}$$

The multihead attention concatenates multiple single attention heads with an output reshaping. A single attention head is computed via:

$$\text{Attn}(\mathbf{H}^l) := \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\mathbf{V}) \tag{2.10}$$

where matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are linear projections of $\mathbf{H}^l$, d_k is the dimension of $\mathbf{Q}$ and $\mathbf{K}$, and softmax is applied row-wise.

A fully-connected feed-forward network is equivalent to a complete graph with pairwise communication. Therefore, Graph Transformers operate over a fully-connected graph to capture global and long-range interactions, addressing the over-smoothing and over-squashing issues from sparse graphs [34]. However, not using the input graph also means that Graph Transformers lose the structural information, only retaining the number of nodes. Therefore, the expressive power of using a fully-connected graph is equivalent to DeepSets, [41] where no edge presents. To restore the topological awareness on the input graph, it is critical for Graph Transformers to equip with PEs to improve their expressivity while maintaining the advantages from the Transformer layers.

2.3.2 Spectral Attention Network (SAN)

SAN [13] is one popular Graph Transformer that uses learnt PEs by applying a Transformer over the eigenfunctions of each node. Its main Transformer layer employs a multi-head attention over all nodes:

$$\mathbf{h}_v^{l+1} = \mathbf{O}_h^l \Big\|_{k=1}^{H} \left(\sum_{u \in V} w_{vu}^{k,l} \mathbf{V}^{k,l} \mathbf{h}_u^l \right) \tag{2.11}$$

where H is the number of heads, l is the layer number, $\Big\|$ is concatenation of multiple single-head attention, and $\mathbf{O}_h^l, \mathbf{V}^{k,l}$ are learnable weights. Additionally, it applies separate

attention mechanisms depending on whether an edge exists in the original graph:

$$\hat{\mathbf{w}}_{uv}^{k,l} = \begin{cases} \left(\mathbf{Q}^{1,k,l}\mathbf{h}_v \circ \mathbf{K}^{1,k,l}\mathbf{h}_j^l \circ \mathbf{E}^{1,k,l}\mathbf{e}_{vu}\right)/\sqrt{d_k} & \text{if } v \text{ and } u \text{ are connected in sparse graph} \\ \left(\mathbf{Q}^{2,k,l}\mathbf{h}_v \circ \mathbf{K}^{2,k,l}\mathbf{h}_j^l \circ \mathbf{E}^{2,k,l}\mathbf{e}_{vu}\right)/\sqrt{d_k} & \text{otherwise} \end{cases} \tag{2.12}$$

$$w_{vn}^{k,l} = \begin{cases} \frac{1}{1+\gamma} \cdot \text{softmax}(\sum_{d_k} \hat{\mathbf{w}}_{vu}^{k,l}) & \text{if } v \text{ and } u \text{ are connected in sprase graph} \\ \frac{\gamma}{1+\gamma} \cdot \text{softmax}(\sum_{d_k} \hat{\mathbf{w}}_{vu}^{k,l}) & \text{otherwise} \end{cases} \tag{2.13}$$

SAN was the first fully-connected Graph Transformer that performed well on benchmarking graph datasets [13]. Its strength lies in its awareness of the Laplace spectrum of the input graph through learnt PEs. These PEs are initialised with the graph Laplacian, which makes the architecture suitable for testing our sheaf-based PEs.

2.4 Related works

Positional information is crucial in many learning tasks. For example, in natural language processing, tokens in an input sentence have sequential ordering. The semantic meaning of the sentence may change when the positions of some tokens are swapped. Recurrent Neural Networks [42] leverage the positional information by sequentially processing the inputs. On the other hand, Transformers [10] treat each token as independent to apply the attention mechanism, requiring PEs to explicitly capture their positions in the input sequence. Similarly, in computer vision, filters in Convolutional Neural Networks [15] are found to implicitly encode the absolute positions of pixels in an image [43].

In GNNs, such positional indexing becomes non-trivial due to the absence of a canonical order within the nodes, making it challenging to define PEs for GNNs. Current approaches either append PEs to node or edge features, and can be categorised according to the positional information they capture [39].

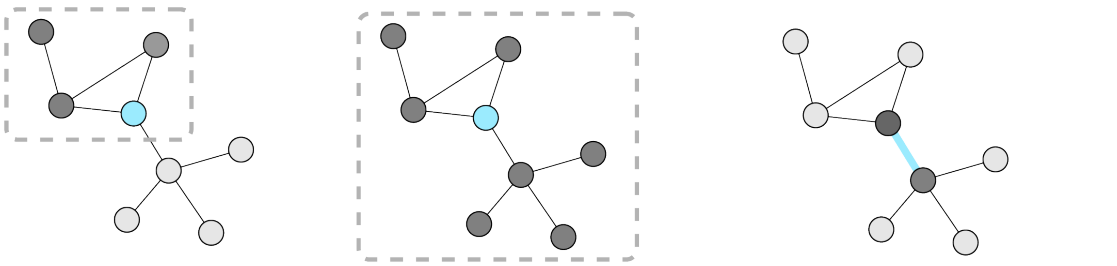


Figure 2.4: Three types of positional encodings.

2.4.1 Local positional encodings

PEs can be defined locally within a node cluster, reflecting the closeness between these nodes. For example, we can compute the distance to the centroid of a cluster, or the sum of each column in a random walk matrix. However, local PEs are still constrained by the notion of locality, only increasing the diameter of the neighbourhood than the 1-hop pairwise communication in message-passing.

2.4.2 Global positional encodings

On the other hand, global PEs capture the node position in the whole graph, breaking the locality constraint with such global knowledge.

Unique indexing Earlier PEs assign a unique identifier to each node via indexing [44]. The arbitrary choice of assignment requires the network to learn all $n!$ permutations. Therefore, the index assignment is sampled from a uniform distribution. Furthermore, although unique indexing differentiates nodes globally, it does not capture any meaningful structural information from the graph.

Graph Laplacians The graph Laplacian provides an effective method to compute PEs for graphs. The graph Laplacian can be constructed using the degree and adjacency matrices. Its eigendecomposition is a spectral technique that embeds the graph into a Euclidean space, providing a unique PE for each node. However, the graph Laplacian PEs are purely structural and suffer from sign ambiguity [12, 13, 45], spurring a number of works to address these issues.

In [46], they propose to treat the graph Laplacian PEs as the solution to a minimisation of dissimilarity between positional embedding and graph structure. They design a more generalised form of such dissimilarity function to go beyond the 2-norm in the graph Laplacian. On the other hand, by decoupling structural and positional representations, [47] proposes to learn the PEs combined with the structural GNNs. This method aims to construct more expressive PEs for arbitrary graphs. Similarly, [48] uses separate channels to update node features and PEs, but with a focus on maintaining the permutation equivariance for more expressive positional features.

However, eigendecomposition comes with sign ambiguity, requiring the model to learn an exponential number of sign possibilities. The above proposals mostly randomly flip the sign of the eigenvectors during training to approximate sign invariance. To tackle this problem, [49] proposes SignNet, which is dedicated to learning the sign invariance for PEs.

2.4.3 Relative positional encodings

Both local and global PEs are appended to the node features as inputs. An alternative method computes relative PEs between two nodes, adding to their edge features. Relative PEs capture the notion of distances or directional relationships between two nodes.

Shortest path One way to represent the relative distance between two nodes is using the shortest path that connects them. However, a complete shortest-path matrix can be computationally expensive. In [32], they propose to use a set of randomly sampled anchor nodes to compute the shortest paths. Additionally, their method learns a distance-weighted aggregation scheme over the anchor nodes to make them more generalisable. Later, [37] proposes a Spatial Encoding that uses the shortest path to measure the spatial relation between nodes. They further associate the encoding with a learnable scalar to integrate it with a Transformer layer.

Random walk matrix Another line of relative PEs utilises the random walk matrix. In [33], they propose Distance Encodings to learn the structural representation of sub-structures, instead of the whole graph, for efficiency. PEs are defined by the distance from the substructure to all other nodes, calculated based on the landing probability of random walks. To reduce the complexity, [47] extends the design by only considering the landing probability of a node to itself, while maintaining its effectiveness. A random walk matrix does not suffer from the sign ambiguity problem of the graph Laplacian matrix.

Diffusion kernels Heat kernel matrix is another metric to measure the relative distance between the nodes. At each timestep, a heat kernel matrix represents the amount of heat transferred from one node to another, simulating a heat diffusion process. Such concept has been utilised in designing relative PEs, such as [50, 51].

2.4.4 Structural Encodings

Apart from PEs, there exists a concept of structural encodings [39]. Rather than representing the positional information of each node, structural encodings represent the structure of a graph or its substructures. However, the design of structural encodings is closely related to its positional counterpart. For example, we can take the eigenvalues, instead of the eigenvectors, of the graph Laplacian to be appended as a global structural encoding for a graph. Structural encodings can also be used to construct PEs, such as counting the number of triangles which a node belongs to. Similarly, structural encodings can be categorised as local, global, or relative, depending on the encoded structure scope.

2.5 Summary

This project focuses on designing a novel form of *global PEs* to improve the expressive power of GNNs in identifying substructures and distinguishing non-isomorphic graphs. As discussed in Section 2.4, PEs can be a computationally efficient way to counter the locality limitation suffered by GNNs by introducing global knowledge of the graph. Furthermore, PEs are also essential to the effectiveness of Graph Transformers, as elaborated in Section 2.3.

More specifically, we aim to extend the graph Laplacian PEs to incorporate semantic information from the nodes, in addition to the structural information of the graph. Knowledge of the node features can be valuable to break symmetries when two nodes are structurally isomorphic. To achieve this goal, we borrow a mathematical tool from algebraic topology, namely a *sheaf*. The sheaf Laplacian is a more generalised form of the graph Laplacian. While the graph Laplacian only encodes the graph structure, the sheaf Laplacian has the ability to take the node data into account, enabling it to represent more complex relationships between the nodes.

Finally, we motivate our design by relating it with other types of PEs and structural encodings. Local PEs can be considered a special form of global PEs by taking the substructure rather than the entire graph. Moreover, if we use the gradient of eigenvectors from sheaf-based PEs, we can obtain relative PEs to augment the edge features. Similarly, we can build structural encodings with sheaf-based PEs by taking the eigenvalues instead of the eigenvectors. Therefore, while our focus lies on building global PEs, the potential impact of our idea can be easily extended to other domains.

Chapter 3

Background: Sheaf Theory

Since we are constructing PEs using a sheaf, we now explain the sheaf theory in more depth. We first introduce the concept of a cellular sheaf (Section 3.1). Moving forward, we describe the computation of the sheaf Laplacian (Section 3.2), which is used to construct sheaf-based PEs. Lastly, we offer a review of existing works that incorporate sheaf theory in GNNs (Section 3.3), motivating the significance of sheaf in the graph learning context.

3.1 Cellular sheaf

In algebraic topology, a cellular sheaf [52, 53] is a mathematical object associated with a graph by attaching vector spaces to nodes and edges. It also defines a map between these vector spaces for each incident node-edge pair, specifying the consistency constraints of the data.

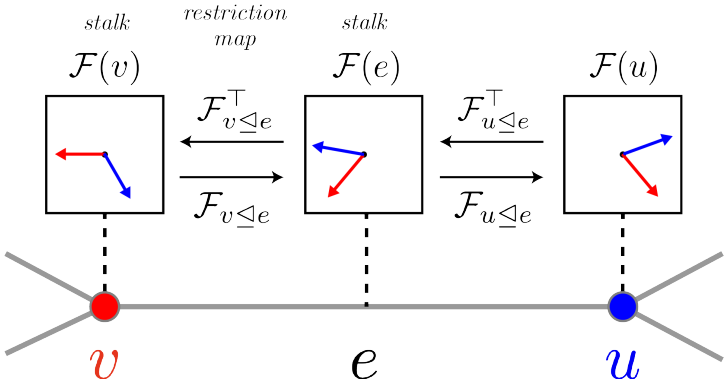


Figure 3.1: A sheaf $(G, \mathcal{F})$ for an edge e connecting two nodes v and u. It associates each node and edge with a vector space ($\mathcal{F}(v)$ and $\mathcal{F}(e)$) and defines a linear map between them ($\mathcal{F}_{v \trianglelefteq e}$). Figure adapted from [54].

Definition 2 (Cellular sheaf) *A cellular sheaf $(G, \mathcal{F})$ on an undirected graph $G = (V, E)$ consists of:*

- *A vector space $\mathcal{F}(v)$ for each vertex $v \in V$.*

- *A vector space $\mathcal{F}(e)$ for each edge $e \in E$.*
- *A linear map $\mathcal{F}_{v \trianglelefteq e} : \mathcal{F}(v) \to \mathcal{F}(e)$ for each incident node-edge pair $v \trianglelefteq e$.*

The vector spaces of the nodes and edges are called *stalks*, while the linear maps for the node-edge pairs are called *restriction maps*. The stalks of nodes and edges form the 0- and 1-cochain, respectively.

Definition 3 (0-cochain) *The space of 0-cochains $C^0(G;\mathcal{F}) := \bigoplus_{v \in V} \mathcal{F}(v)$ is the space formed by all the stalks associated to the nodes of the graph, where $\bigoplus$ denotes the direct sum of vector spaces.*

Definition 4 (1-cochain) *The space of 1-cochains $C^1(G;\mathcal{F}) := \bigoplus_{e \in E} \mathcal{F}(e)$ is the space formed by all the stalks associated to the edges of the graph.*

These two cochains lead to the definition of a co-boundary map $\delta : C^0(G,\mathcal{F}) \to C^1(G,\mathcal{F})$. The co-boundary map δ measures the *inconsistency* between all the nodes. For two nodes $v, u \in V$, their node data $\mathbf{x}_v \in \mathcal{F}(v)$ and $\mathbf{x}_u \in \mathcal{F}(u)$ are considered as *consistent* over an edge $e \in E$ if $\mathcal{F}_{v \trianglelefteq e}\mathbf{x}_v = \mathcal{F}_{u \trianglelefteq e}\mathbf{x}_u$.

Definition 5 (Co-boundary map) *Given an arbitrary orientation of an edge $e = u \to v$, the co-boundary map $\delta : C^0(G,\mathcal{F}) \to C^1(G,\mathcal{F})$ is defined as $(\delta\mathbf{x})_e = \mathcal{F}_{v \trianglelefteq e}\mathbf{x}_v - \mathcal{F}_{u \trianglelefteq e}\mathbf{x}_u$, where $\mathbf{x}$ is a block-vector stacking all node data $\mathbf{x}_v$ for all $v \in V$.*

We now provide an intuitive interpretation from the perspective of opinion dynamics, as proposed in [55]. The node stalk $\mathcal{F}(v)$ can be seen as the space of "private opinions" for a node v, where the node data $\mathbf{x}_v \in \mathcal{F}(v)$ is the value of an opinion. The edge stalk $\mathcal{F}(e)$ forms a public "discourse space" shared by node opinions. Therefore, the restriction map $\mathcal{F}_{v \trianglelefteq e}$ describes how a node opinion publicly manifests in such a discourse space. Furthermore, the space of 0-cochain is formed by all private opinions, and the space of 1-cochain is formed by all discourse spaces. Since the co-boundary map δ maps between the two co-chains, it measures the disagreement between all the nodes.

3.2 Sheaf Laplacian

We can construct the sheaf Laplacian using the co-boundary map δ. The sheaf Laplacian operator for a given cellular sheaf measures the aggregated "disagreement of opinions" at each node. Formally, it can be defined as:

Definition 6 (Sheaf Laplacian) *Given a cellular sheaf $(G;\mathcal{F})$, the sheaf Laplacian is a linear map $L_{\mathcal{F}} : C^0(G,\mathcal{F}) \to C^0(G,\mathcal{F})$. It can be constructed using the co-boundary maps $L_{\mathcal{F}} = \delta^\top\delta$, or defined node-wise $L_{\mathcal{F}}(\mathbf{x})_v = \sum_{v,u \trianglelefteq e}(\mathcal{F}_{v \trianglelefteq e}\mathbf{x}_v - \mathcal{F}_{u \trianglelefteq e}\mathbf{x}_u)$.*

Given a graph G with n nodes, assuming all stalks have a fixed dimension d, the sheaf Laplacian is an $nd \times nd$ positive semi-definite block matrix. It has diagonal blocks

$L_{\mathcal{F}_{vv}} = \sum_{v \trianglelefteq e} \mathcal{F}^{\top}_{v \trianglelefteq e} \mathcal{F}_{v \trianglelefteq e}$, and non-diagonal blocks $L_{\mathcal{F}_{vu}} = -\mathcal{F}^{\top}_{v \trianglelefteq e} \mathcal{F}_{u \trianglelefteq e}$. The normalised sheaf Laplacian becomes $\Delta_{\mathcal{F}} := \mathbf{D}^{-1/2} L_{\mathcal{F}} \mathbf{D}^{-1/2}$, where $\mathbf{D}$ is the block-diagonal of $L_{\mathcal{F}}$.

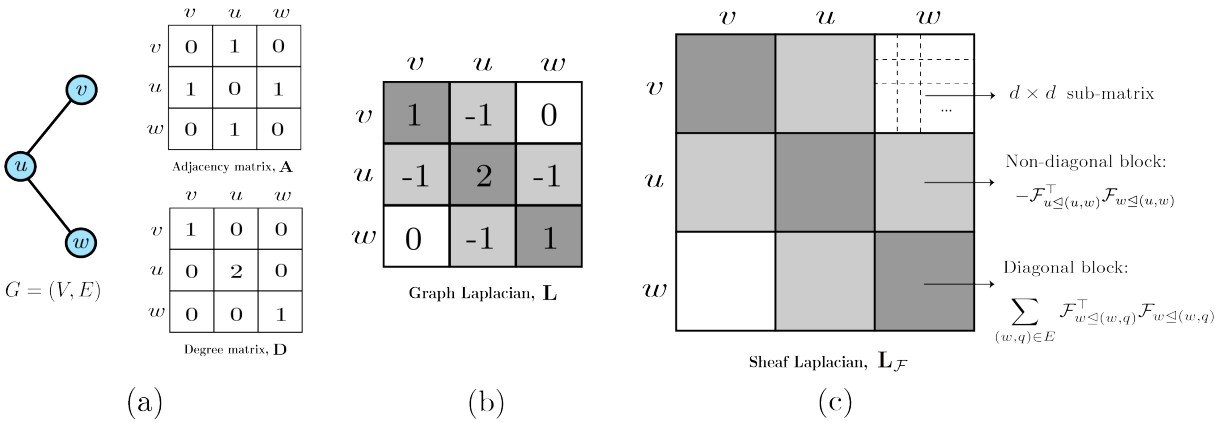


Figure 3.2: The graph Laplacian matrix (b) and the sheaf Laplacian matrix (c) for a given graph as illustrated in (a).

Sheaf Laplacian generalises graph Laplacian The graph Laplacian is defined as $L = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix and $\mathbf{A}$ is the adjacency matrix of the graph. In a similar fashion as we define sheaf Laplacian using co-boundary maps, we can construct graph Laplacian using the incidence matrix.

Definition 7 (Incidence matrx) *Given an arbitrary choice of orientation of an edge $e = u \to v$, the incidence matrix $\boldsymbol{B}$ is defined as:*

$$\boldsymbol{B}_{xe} = \begin{cases} 1, & \text{if } x = u \\ -1, & \text{if } x = v \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

The graph Laplacian can then be constructed by $L = \mathbf{B}^{\top}\mathbf{B}$. Unlike how the co-boundary map (Definition 5) uses the node data $\mathbf{x}_v$, the incidence matrix is built purely on adjacency information (whether and how a node x belongs to a arbitrarily directed edge e). In fact, the graph Laplacian is a trivial sheaf, where all the stalks are set to scalars ($d = 1$, where d is the stalk dimension) and the restriction maps are identity functions. The sheaf Laplacian generalises the graph Laplacian by allowing a higher dimension ($d \geq 1$). This gives a sheaf the potential to encode richer information, such as the semantic knowledge of the node data. Furthermore, the sheaf Laplacian may capture more general node relationships as the stalk dimension increases.

3.3 Sheaf theory in GNNs

Prior works [55, 54, 56] focused on analysing the underlying geometry of the graph using sheaf theory, giving rise to Sheaf Neural Networks [55]. In [55, 54], they model the

convolution of GNNs as a heat diffusion process which implicitly uses the graph Laplacian, and generalise the diffusion to operate over a cellular sheaf. Since our project focuses on a different domain, we leave the detailed description of Sheaf Neural Networks in Appendix A. Here, we provide a high-level intuition behind this idea. Given a node representation matrix $\mathbf{H} \in \mathrm{R}^{n \times d_h}$, a Graph Convolution Network [1] updates the node features via:

$$\mathbf{H}_{l+1} = \sigma((\mathbf{I} - \Delta_0)\mathbf{H}_l\mathbf{W}) \tag{3.2}$$

where l is the layer number, Δ_0 is the normalised graph Laplacian, $\mathbf{I}$ is the identity matrix, and $\mathbf{W}$ is the weight matrix with σ as a non-linear activation function. This equation comes from the Euler discretisation of a heat diffusion process to model the convolution operator of GNNs.

Sheaf Neural Networks replace the graph Laplacian Δ_0 in Equation A.2 with the sheaf Laplacian $\Delta_{\mathcal{F}}$, leading to a sheaf diffusion process. The sheaf Laplacian is more general than the graph Laplacian as explained in Section 3.2, encoding a richer relational structure of the graph. Empirically, these works [55, 54, 56] show that a sheaf diffusion process improves the performance of GNNs, especially on heterophilic graphs. While [55] hand-crafts the sheaf, [54] proposes a pipeline to learn the sheaf directly from the data. Following up, [56] provides an alternative method to pre-compute the sheaf by analogy to parallel transport on a manifold, avoiding the numerical issues from gradient-based learning.

Our project applies the sheaf theory to a different domain. Unlike Sheaf Neural Networks, we do not modify the GNN architecture. Instead, we construct positional encodings using the sheaf to encode the graph structure. We aim to further motivate the paradigm shift in the current perspective on spectral GNNs, which was primarily built upon the graph Laplacian. Furthermore, our project also contributes to designing novel GNN techniques using concepts from geometry and topology, inspiring more investigation in this interconnected area.

Chapter 4

Implementation

The framework in Figure 4.1 outlines the implementation details elaborated in this chapter. We first describe the construction of Laplacian-based PEs, using the graph Laplacian as an example (Section 4.1). Next, we explain our sheaf-based PEs using a pre-computed sheaf (Section 4.2) and a learnt sheaf (Section 4.3). Finally, we address the sign invariance problem associated with eigendecomposition (Section 4.4).

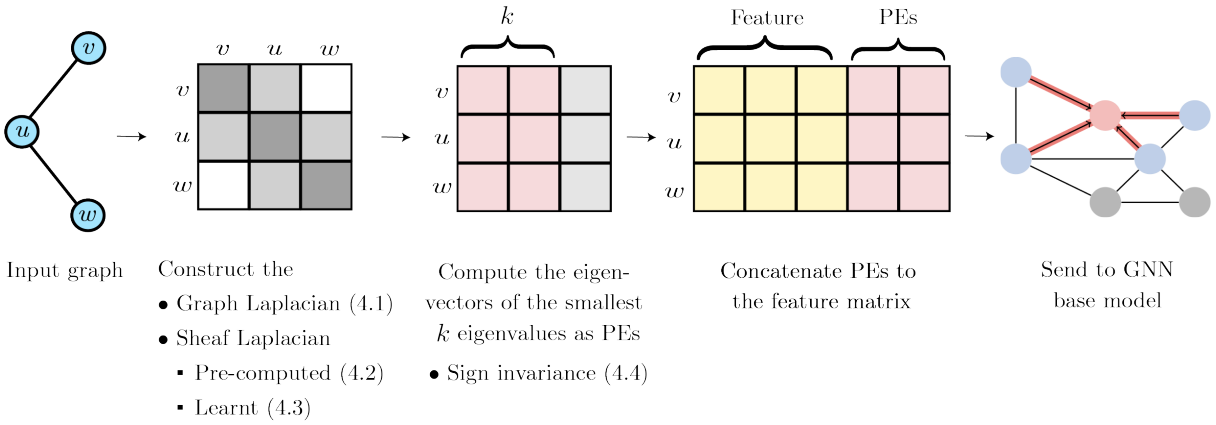


Figure 4.1: The framework of our implementation. Given an input graph, we construct the graph/sheaf Laplacian matrix and take the k eigenvectors with the smallest eigenvalues as PEs. The PEs are concatenated to the node feature matrix as inputs to the GNN. This chapter focuses on explaining the three PE construction methods (using the graph Laplacian, the pre-computed sheaf Laplacian, and the learnt sheaf Laplacian).

4.1 Laplacian-based PEs

Our proposal using the sheaf Laplacian extends the graph Laplacian, both constructing PEs with the Laplacian matrix. The graph Laplacian PEs are also used as the baseline for evaluation. We first motivate using *Laplacian eigenvectors* to capture global structural information of the graph, and then explain how they are implemented as PEs in practice.

The Laplacian eigenvectors have been employed in unsupervised manifold learning [11].

The spectral decomposition of the Laplacian matrix computes a low-dimensional embedding of the data on a manifold. More importantly, such an embedding optimally preserves the graph structure. The Laplacian matrix is also widely used in graph signal processing [57] and spectral GNNs [58]. These desirable properties make the Laplacian matrix an excellent candidate for constructing PEs [12, 13, 47].

Graph Laplacian Let us take the graph Laplacian L to illustrate the idea. The graph Laplacian can be computed via:

$$L = \mathbf{D} - \mathbf{A} \tag{4.1}$$

where $\mathbf{D}$ is the degree matrix and $\mathbf{A}$ is the adjacency matrix of the graph. We compute the eigendecomposition of the graph Laplacian:

$$L = \mathbf{U}^T \mathbf{\Lambda} \mathbf{U} \tag{4.2}$$

Here, $\mathbf{\Lambda}$ denotes the n eigenvalues and $\mathbf{U}$ denotes the $n \times n$ eigenvector matrix for a graph with n nodes. Such factorisation of the graph Laplacian forms a meaningful local coordinate system which respects the graph structure. Therefore, we can use the eigenvectors as PEs to provide unique representations for the graph nodes.

As illustrated in Figure 4.1, we stack input node features as a matrix $\mathbf{X} \in \mathbb{R}^{n \times d_x}$, where n is the number of nodes and d_x is the feature dimension. To append PEs to each node, we take the eigenvectors corresponding to the smallest k eigenvalues from $\mathbf{U}$, and concatenate them with $\mathbf{X}$. This results in a new feature matrix $\mathbf{X}' \in \mathbb{R}^{n \times (d_x + k)}$, which are then fed into the GNN. The advantage of such an approach is that PEs can be computed at preprocessing time. Furthermore, it is computationally efficient. The complexity for complete factorisation is $O(E^{3/2})$. If an approximation method such as the Nystrom method [59] is used, the complexity reduces to $O(n)$.

4.2 Pre-computed connection Laplacian

The graph Laplacian only encodes the graph structure, irrespective of the node features. As explained in Chapter 3, the sheaf Laplacian generalises the graph Laplacian with an additional parameterisation from the node data, which gives it the potential to construct more expressive PEs. This section elaborates on our method to construct sheaf-based PEs by pre-computing the *connection Laplacian* [53, 56].

4.2.1 Manifold assumption

The method is built upon the manifold assumption, stating that although data points live in a high-dimensional ambient space $\mathbb{R}^p$, they can be embedded onto a low-dimensional

Riemannian manifold $\mathcal{M}^q$, where $q \ll p$. The key objective of such embedding is to capture the geometric and topological structure of the high-dimensional data.

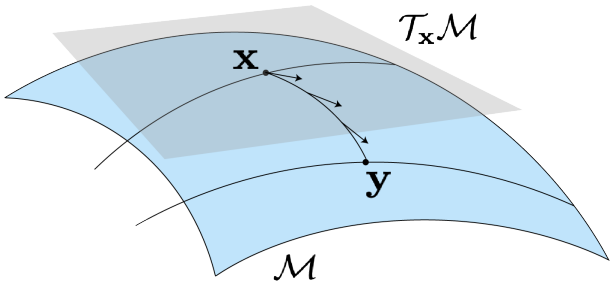


Figure 4.2: An illustration of a Riemannian manifold.

As shown in Figure 4.2, a *Riemannian manifold* $\mathcal{M}$ generalises Euclidean geometry such that the surfaces are not necessarily flat. Each point on the manifold $\mathbf{x} \in \mathcal{M}$ is associated with a *tangent space* $\mathcal{T}_{\mathbf{x}}\mathcal{M}$, which is a Euclidean space. A tangent space is a vector space that contains all possible tangential directions passing through the point $\mathbf{x}$. Intuitively, it generalises the notion of tangent lines for curves from two dimensions to higher dimensions. Next, *parallel transport* denotes a mechanism of transporting points on a manifold along smooth curves, generalising the notion of translation in Euclidean space.

4.2.2 Analogy between sheaf and parallel transport

Based on the manifold assumption, we can compute the sheaf Laplacian by an analogy to the parallel transport on a manifold. This is because when we constrain the restriction maps to be an orthogonal matrix, we obtain a special form of the sheaf Laplacian, called the *connection Laplacian* [53]. A connection Laplacian can be seen as a discretised version of the *vector bundle*, which is an object from differential geometry that associates a vector space to each point on a manifold, similar to how sheaf attaches vector spaces (stalks) to nodes and edges. Therefore, we can observe the link between the connection Laplacian and a manifold. While the restriction maps $\mathcal{F}_{v \trianglelefteq e}$ describe how vectors are transported between stalks for a sheaf, parallel transport is a way to transport vectors on a manifold.

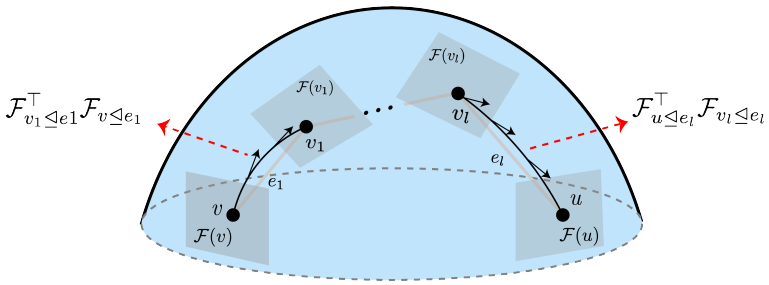


Figure 4.3: Analogy between transport on a sheaf and parallel transport on a manifold.

As illustrated in Figure 4.3, we define the notion of *transport* for a sheaf $\mathbf{P}^{\gamma}_{v \to u} : \mathcal{F}(v) \to \mathcal{F}(u)$ from the stalk $\mathcal{F}(v)$ to the stalk $\mathcal{F}(u)$ by composing restriction maps along the edges:

$$\mathbf{P}^{\gamma}_{v\to u} = (\mathcal{F}^{\top}_{u\trianglelefteq e_l}\mathcal{F}_{v_l\trianglelefteq e_l})...(\mathcal{F}^{\top}_{v_1\trianglelefteq e_1}\mathcal{F}_{v\trianglelefteq e_1}) \tag{4.3}$$

where $v, u \in V$ are two nodes in the graph, and $\gamma_{v\to u} = (v, v_1, ..., v_l, u)$ denotes the sequence of nodes in the path from v to u.

We can then compute a connection Laplacian using the parallel transport on the manifold. For two nodes v and u, its transport maps $\mathbf{P}^{\gamma}_{v\to u}$ from $\mathcal{F}(v)$ to $\mathcal{F}(u)$ corresponds to the parallel transport from $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ to $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$, where $\mathbf{x}_v$ and $\mathbf{x}_u$ are the associated node data. When v and u are connected by an edge e, we derive from Equation 4.3 that the transport $\mathbf{P} = \mathbf{P}^{\gamma}_{v\to u} = \mathcal{F}^{\top}_{u\trianglelefteq e}\mathcal{F}_{v\trianglelefteq e}$. As explained in Section 3.2, we can now compute the non-diagonal blocks of the sheaf Laplacian using $L_{\mathcal{F}_{vu}} = -\mathcal{F}^{\top}_{v\trianglelefteq e}\mathcal{F}_{u\trianglelefteq e}$. Furthermore, since the connection Laplacian is an orthogonal matrix satisfying $L_{\mathcal{F}}L^{\top}_{\mathcal{F}} = \mathbf{I}$, we can compute the diagonal blocks by taking the degree of the nodes $L_{\mathcal{F}_{vv}} = \text{degree}(v)$. In this way, we fill up all the entries in the connection Laplacian matrix. We then follow the same procedure as described in Section 4.1, where we take the first k eigenvectors of the connection Laplacian as PEs followed by a reshape to concatenate them with the node feature matrix.

4.2.3 Compute parallel transport on the manifold

The last question is how we compute the parallel transport from $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ to $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$ on the manifold $\mathcal{M}$ in the context of graph data. In [56], they propose a method to pre-compute this transport for Sheaf Neural Networks [55]. Here, we explore its application to construct sheaf-based PEs.

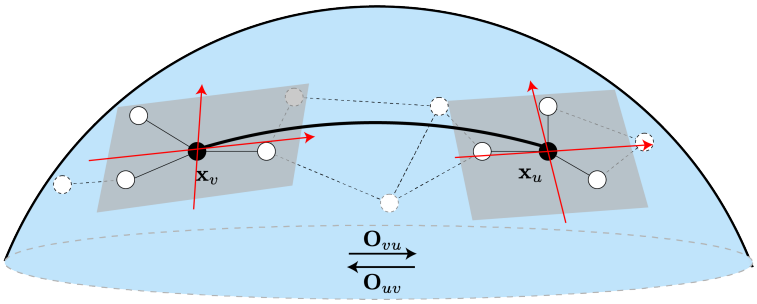


Figure 4.4: Step 1: We construct the orthonormal bases of $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ and $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$ using local PCA, defined by the 1-hop neighbourhood of the nodes. Step 2: We compute the orthogonal mapping $\mathbf{O}_{vu}$ by optimal aligning the two bases, which approximates the parallel transport operator [53].

The two-step procedure is summarised in Figure 4.4. Here, we elaborate the steps in more details. First, we construct the orthonormal bases of the tangent spaces $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ for each data point $\mathbf{x}_v$ via local Principle Component Analysis (PCA). The local neighbourhood is defined by the 1-hop neighbourhood $\mathcal{N}(v)$ of the node v, giving us a $p \times |\mathcal{N}(v)|$ matrix $\hat{\mathbf{X}}_v = [\mathbf{x}_v - \mathbf{x}_v, ..., \mathbf{x}_{v_{|\mathcal{N}(v)|}} - \mathbf{x}_v]$. We perform PCA via Singular Value Decomposition (SVD) on the matrix $\hat{\mathbf{X}}_v = \mathbf{U}_v\sum_i\mathbf{V}^{\top}_v$. The stalk dimension d is set as a hyperparameter,

which is also the dimension of the orthonormal bases. Therefore, we take the first d left singular vectors of $\mathbf{U}_v$. This forms a d-dimensional subspace of $\mathbb{R}^p$, calling it $\mathbf{O}_v$. We take $\mathbf{O}_v$ as an approximation for the basis of the tangent space $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$.

Next, we compute the parallel transport via optimal alignment between the tangent spaces $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ and $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$. Intuitively, an optimal alignment is a mapping from one tangent space to another. Formally, we compute $\mathbf{O}_{vu} = \mathbf{U}\mathbf{V}^\top$, where $\mathbf{U}$ and $\mathbf{V}$ comes from the SVD of $\mathbf{O}_v^\top \mathbf{O}_u = \mathbf{U}\sum\mathbf{V}^\top$. If $\mathbf{x}_v$ and $\mathbf{x}_u$ are close enough, then $\mathbf{O}_{ij}$ approximates the parallel transport between their tangent spaces as proved in [53]. This condition is ensured by taking the 1-hop neighbourhood as explained in the last paragraph. Note when there are fewer than d neighbours for node v, we take the nearest nodes according to the Euclidean distances of their node data.

4.2.4 Learnable pre-computed PEs

The pre-computed connection Laplacian is computationally favourable. However, not all input node features are informative enough to construct it. For example, some datasets only provide scalar node features, challenging the manifold assumption where high-dimensional data is expected.

To tackle this problem, we allow the PEs to evolve after initialising them with the pre-computed sheaf. Learnable structural and positional encodings (LSPE) was explored in [47], but they only applied the technique to the graph Laplacian and random walk matrix. We investigate this mechanism under the sheaf Laplacian context. Furthermore, the motivation is different. Firstly, we complement the pre-computed connection Laplacian PEs with an evolvable structure to address the issue when the manifold assumption does not hold. Secondly, the underlying sheaf also evolves as node representations get updated after each layer. Therefore, our pre-computed connection Laplacian PEs may enjoy additional benefits from learning.

At initialisation, for each node $v \in V$, we embed the pre-computed sheaf-based PEs into the same dimension as the node features:

$$\mathbf{h}_v^0 = \mathbf{x}_v \tag{4.4}$$

$$\mathbf{p}_v^0 = \mathrm{MLP}(\mathbf{p}_v^{\mathrm{ConnLap}}) \tag{4.5}$$

where $\mathbf{h}_v$ is the node representation, $\mathbf{x}_v$ is the input node feature, $\mathbf{p}_v$ is its PE, and $\mathbf{p}_v^{\mathrm{ConnLap}}$ is the initialised PE with the connection Laplacian. During training, we update both node and positional representations:

$$\mathbf{h}_v^{l+1} = f_h([\mathbf{h}_v^l || \mathbf{p}_v^l], \{[\mathbf{h}_u^l || \mathbf{p}_u^l]\}_{u \in \mathcal{N}(v)}) \tag{4.6}$$

$$\mathbf{p}_v^{l+1} = f_p(\mathbf{p}_v^l, \{\mathbf{p}_u^l\}_{u \in \mathcal{N}(v)}) \tag{4.7}$$

where l is the layer number, $||$ denotes concatenation, f_h is the update function for node representations depending on the GNN layer used, and f_p swaps the activation function in f_h to `tanh` to accommodate both positive and negative values in positional coordinates.

4.3 Learnt sheaf Laplacian

However, the pre-computed connection Laplacian may not be general enough to capture the underlying sheaf of the graph truthfully. Furthermore, the learning mechanism we augmented to let it evolve is not using the sheaf structure as signals. Hence, we now consider a method to learn the sheaf directly from the data, in order to construct more general PEs to better suit the graph. In [54], they propose an end-to-end model to learn a sheaf diffusion process for GNNs. We adopt their framework by learning the restriction maps with a parametric function, and apply it to construct sheaf-based PEs.

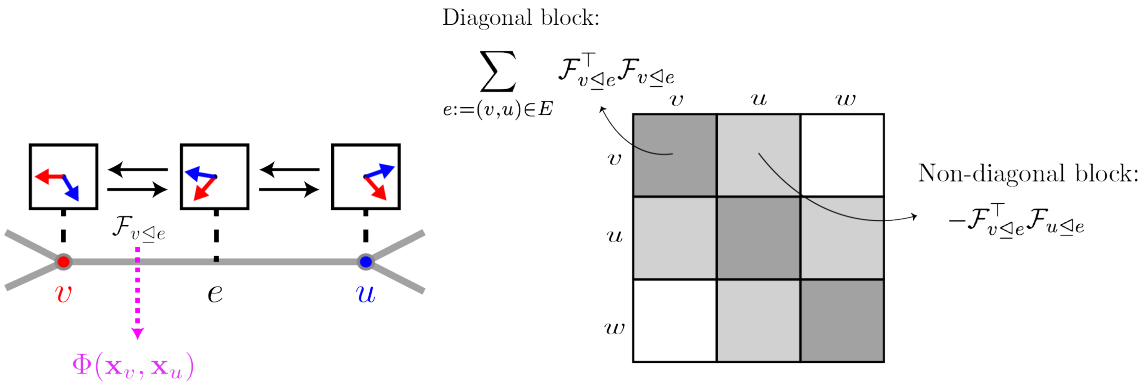


Figure 4.5: (a) We use a parametric function $\Phi(\mathbf{x}_v, \mathbf{x}_u)$ to learn the restriction maps. (b) We then construct the sheaf Laplacian by filling up diagonal and non-diagonal blocks. Different types of sheaf can be learnt depending on the type of function Φ.

4.3.1 Learning the restriction maps

As shown in Figure 4.5, the restriction maps $\mathcal{F}_{v \trianglelefteq e} : \mathcal{F}(v) \to \mathcal{F}(e)$ are linear maps between the stalks associated with node v and edge e, where v and e is an incident node-edge pair. The restriction map $\mathcal{F}_{v \trianglelefteq e}$ is a $d \times d$ matrix, where d is a tunable hyperparameter denoting the dimension of the stalks. The idea is to approximate the restriction map using a learnable parametric function $\Phi : \mathbb{R}^{d \times 2} \to \mathbb{R}^{d \times d}$:

$$\mathcal{F}_{v \trianglelefteq e:=(v,u)} = \Phi(\mathbf{x}_v, \mathbf{x}_u) \tag{4.8}$$

where $\mathbf{x}_v$ and $\mathbf{x}_u$ are the node features of nodes v and u. Since the restriction maps denote the consistency constraints of the data as explained in Section 3.1, we feed in the node

features to learn these maps.

In practice, the parametric function Φ is implemented as an MLP that takes the concatenation of two node features as the input. The output is then reshaped to fit the size of the restriction maps:

$$\Phi(\mathbf{x}_v, \mathbf{x}_u) = \sigma(\mathbf{W}[\mathbf{x}_v || \mathbf{x}_u] + \mathbf{b}) \tag{4.9}$$

where $||$ denotes matrix concatenation, $\mathbf{W}$ is the weight matrix for the MLP with an optional bias $\mathbf{b}$, and σ is a non-linear activation function. In [54], they show that if Φ has enough capacity and the features are diverse enough, it would allow us to learn any type of sheaf over a graph.

4.3.2 Types of sheaf learnt

Depending on the type of function Φ, we learn a sheaf Laplacian with a corresponding matrix type. These sheaf types come with different trade-offs, and we will empirically analyse their effects when constructing sheaf-based PEs.

Diagonal The sheaf Laplacian with diagonal blocks is the most data efficient due to the fewer parameters to learn. It is also computationally cheaper if we perform sparse matrix multiplications. However, its generality to capture different sheaves is also limited due to the small number of parameters.

Orthogonal Following [54], we build the orthogonal matrix using the Householder reflections [60]. This type of sheaf is the same as the connection sheaf explained in Section 4.2. It uses more parameters than the diagonal matrix, but still fewer than a general sheaf due to the orthogonality constraint. Furthermore, because of the analogy to parallel transport on a manifold, the connection Laplacian provides better understood geometric properties to interpret.

General Learning a general sheaf puts no constraint on the matrix learnt, and is hence most flexible to represent any possible sheaf of the graph. However, a large number of parameters puts it at the risk of overfitting, making such type of a sheaf harder to train.

4.3.3 Eigendecomposition during training

The difficulty of constructing PEs using learnt sheaf also lies in the eigendecomposition during training. Unlike the pre-computed PEs, we can no longer use both time- and memory-efficient eigensolvers, such as `scipy.linalg.eigsh`, which do not allow gradient-based learning. Furthermore, the LAPACK routine used by common differentiable eigensolvers, such as `torch.linalg.eigsh`, are numerically unstable. In degenerate cases

where repeated eigenvalues are present, the eigensolver produces an infinite gradient. One mitigation technique is introducing a small amount of random perturbation to the (diagonal of) Laplacian matrix, hopefully producing unique eigenvalues. However, this technique adds imprecision and requires non-trivial handling to decide the perturbation magnitude. Our initial experiments show that the performance was significantly affected when perturbation was added.

To alleviate the problem, we use a more advanced algorithm provided by `xitorch` [61]. We use `xitorch.linalg.symeig`, which performs partial eigendecomposition on the smallest k pairs of eigenvalues and eigenvectors with explicit handling on degenerate cases [62]. The algorithm splits the eigenvectors into three components to find parallel, orthogonal, and degenerate parts separately. Moreover, for large-scale sparse matrices, we use the Davidson algorithm [63] implemented by the eigensolver to boost the memory efficiency.

4.4 Sign invariance

The role of symmetry is critical in machine learning to reduce the problem space and improve data efficiency. If a model can capture the underlying symmetry of the problem, then it can better generalise to unseen data after learning those in the same symmetry group. There are nontrivial symmetries in our sheaf-based PEs. When we perform eigendecomposition from the sheaf Laplacian, there is a sign ambiguity problem due to how eigenvectors come in pairs. If $\mathbf{v}$ is an eigenvector, then $-\mathbf{v}$ is also valid. If we normalise the Laplacian matrix, it requires the model to learn all 2^k sign possibilities, where k is the number of eigenvectors used as PEs. Without normalisation, there will be more ambiguities. Ideally, we want a function f that is sign-invariant:

$$f(\mathbf{v}_1, \ldots, \mathbf{v}_k) = f(s_1\mathbf{v}_1, \ldots, s_k\mathbf{v}_k), \quad \text{where } s_i \in \{-1, 1\} \text{ for } i \in \{1, \ldots, k\} \tag{4.10}$$

Therefore, we randomly flip the signs of the eigenvectors during training. This is equivalent to a random uniform sampling from all the 2^k sign possibilities. Since this is an approximation to sign invariance, the choice of k needs to be kept small ($k \ll n$, where n is the number of nodes). Furthermore, it must also be small enough to accommodate small graphs. Nonetheless, random flipping is necessary for the expressive power of Laplacian-based PEs. In [12], they compared random flipping with taking the absolute values of the eigenvectors, and empirically showed the importance of sign invariance for such PEs.

4.5 Summary

In summary, we explain the implementation of three types of PEs, which are the subjects to evaluate in the following chapters.

- **GraphLap:** PEs constructed with the graph Laplacian during preprocessing. It will be the baseline to evaluate our sheaf-based PEs.
- **ConnLap:** PEs constructed with the connection Laplacian during preprocessing. The connection Laplacian is a sheaf Laplacian with an orthogonal matrix. We augment it with an optional learning mechanism to tackle the problem caused by less informative node data.
- **SheafLap:** PEs constructed with a learnt sheaf Laplacian from the node data. We implement three types of matrices associated with the sheaf and discuss the differentiable eigendecomposition method used.

Chapter 5

Evaluation

The goal of evaluation is to test whether our proposed sheaf-based PEs improve the expressive power of GNNs and Graph Transformers. We present our experimental results for node-level tasks (Section 5.1) and graph-level tasks (Section 5.2). This is followed by a qualitiative analysis (Section 5.3) and complexity evaluation (Section 5.4). Our main questions are:

1. Can sheaf-based PEs outperform the graph Laplacian PEs in producing differentiable node embeddings as required by node-level tasks?
2. Can they distinguish more non-isomorphic graphs in graph-level tasks?
3. Are they effective across models, including Graph Transformers?
4. What are the trade-offs between the pre-computed sheaf and the learnt sheaf when constructing sheaf-based PEs?

5.1 Node-level Tasks

Node-level tasks require the model to learn appropriate node representations. However, the message-passing mechanism makes it difficult to differentiate nodes with similar local structures. An ideal positional encoding should not only provide necessary global information of the nodes, but also handle the intricate relationships between them, such as heterophily.

Datasets We use the same set of real-world datasets following [64, 54, 56], covering citation networks [65, 66] (Cora, Citeseer, Pubmed), webpages [67] (Cornell, Texas, Wisconsin), actor co-occurrence networks [68] (Film), and Wikipedia networks [69] (Chameleon, Squirrel). Each of these datasets consists of a single graph. For example, in the Cora dataset, nodes are academic pages with edges representing citation relationships. The goal is to learn appropriate node representations to correctly classify the nodes, such as their

academic topics in the Cora dataset. We append a detailed description for each dataset in Appendix B. This set of datasets contains graphs with varying sizes and densities, as shown in Table 5.1. Furthermore, the graphs are ranked in increasing order of homophily level. A higher homophily level indicates that connected nodes are more likely to have similar features.

	Texas	**Wisconsin**	**Film**	**Squirrel**	**Chameleon**	**Cornell**	**Citeseer**	**Pubmed**	**Cora**
#Nodes	183	251	7,600	5,201	2,277	183	3,327	18,717	2,708
#Edges	295	466	26,752	198,493	31,421	280	4,676	44,327	5,278
#Classess	5	5	5	5	5	5	7	3	6

Table 5.1: Graph statistics for the datasets used in node-level tasks, in the increasing order of homophily level.

Setup We use the 10 fixed splits provided by [70], each containing 48%/32%/20% of nodes per class for training, validation, and testing, respectively. We follow the same hyperparameter settings in [54] for each dataset. For our model-specific parameters, we use the 8 smallest eigenvectors as PEs, and a dimension $d = 3$ for the stalks in the sheaf. We use GCN as the base model, due to its well-known inability to deal with heterophilic graphs. The reported results are the mean and standard deviation of accuracy (%) across the 10 folds.

5.1.1 Laplacian-based PEs

	Texas	**Wisconsin**	**Cornell**
#Nodes	183	251	183
#Edges	295	466	280
No PE	57.30±5.51	49.80±6.80	45.95±6.84
GraphLap	58.22±7.03	55.49±12.46	**51.35±7.15**
ConnLap	58.38±7.76	**57.65±6.63**	**52.97±7.37**
SheafLap(diag)	**61.35±6.63**	54.90±9.80	44.32±7.17
SheafLap(orth)	**61.08±6.19**	54.51±7.22	48.38±5.05
SheafLap(gen)	60.81±7.07	**57.65±5.24**	45.68±6.56

Table 5.2: Mean±std accuracy (%) for node-level tasks with smaller graphs across 10 folds using different types of positional encodings. The best and the second best results are highlighted in **red** and **blue**, respectively.

Effectiveness of PEs As shown in Table 5.2, the use of PEs (GraphLap, ConnLap, SheafLap) improves performance over vanilla GCN (NoPE) in all three datasets. While previous works [12, 47] focused on evaluating graph-level tasks, we also demonstrate the effectiveness of such types of PEs on node-level tasks. Furthermore, the graphs in these datasets are very sparse, exacerbating the locality problem in message-passing. Hence, the global structural knowledge from the Laplacian spectrum is helpful to improve the

expressivity of GNNs in identifying substructures. The performance gain is notable even with the simplest Laplacian (GraphLap).

Sheaf-based PEs outperform GraphLap We also observe that sheaf-based PEs (ConnLap, SheafLap) achieve better performances than the graph Laplacian (GraphLap) in most cases. Furthermore, ConnLap outperforms GraphLap in all tasks, while SheafLap obtains the best results in two datasets except in Cornell, which we will discuss later. The improvement aligns with our motivation that the sheaf Laplacian can additionally incorporate node data to provide a more expressive embedding space, compared to how the graph Laplacian only uses adjacency information. Furthermore, all variants of SheafLap uniformly achieve outstanding results in Texas, which is the most heterophilic dataset here. This suggests that PEs constructed from the sheaf Laplacian can better capture such complex relationships between the nodes, giving rise to more expressive PEs.

5.1.2 Pre-computed connection Laplacian as PEs

In Table 5.2, compared to the learnt sheaf Laplacian (SheafLap), ConnLap is more stable across datasets. This can be explained by the rich information the node features provide, where the feature dimension is high (larger than 1000). Therefore, the manifold assumption can hold in these datasets, where high-dimensional data live on a low-dimensional manifold. In addition, the pre-computed ConnLap also avoids the numerical issues which learnt SheafLap suffers during backpropagation, contributing to its better stability. Another possibility is that these graphs are relatively small, so the amount of data may not be sufficient for the learnt sheaf to be properly trained.

5.1.3 Learnt sheaf Laplacian as PEs

The performance of SheafLap is even worse than not using PEs in the Cornell dataset, as shown in Table 5.1. One possibility is related to the numerical instability from differentiable eigensolver discussed in Section 4.3.3, which can impair the PE quality significantly when the Laplacian matrix is not well-conditioned. There are also other factors that may influence the performance of SheafLap, which we provide a deeper analysis of them in the following experiments.

Effect of sheaf type We compare different types of learnt sheaf Laplacian: diagonal (diag), orthogonal (orth), and general (gen), as described in Section 4.3.2. In Table 5.2, we observe that sheaf type can have very different effects on different datasets. In Cornell, the performance difference is as significant as 4.06%. We cannot conclude which sheaf type is the best, as each type obtains the best result in one of the three datasets. While a diagonal matrix has the fewest parameters to train, a general sheaf is more flexible to capture different types of sheaf. Therefore, we choose to use sheaf with an orthogonal

matrix for the rest of experiments, balancing the trade-off between computational cost and generality among the three matrices.

SheafLap	Texas		Wisconsin		Cornell	
	norm	unnorm	norm	unnorm	norm	unnorm
Diagonal	60.54±6.53	**61.35±6.63**	54.14±7.62	**54.90±9.80**	**44.86±4.55**	44.32±7.17
Orthogonal	59.19±6.89	**61.08±6.19**	54.49±7.44	**54.51±7.22**	46.49±5.90	**48.38±5.05**
General	59.02±7.34	**60.81±7.07**	55.89±6.81	**57.65±5.24**	45.26±4.31	**45.68±6.56**

Table 5.3: Mean±std accuracy (%) for node-level tasks comparing learnt sheaf PEs with normalised and unnormalised sheaf Laplacian. The winning case in each setup is highlighted in **red**.

Effect of normalisation We also compare the normalised and unnormalised sheaf Laplacian for constructing PEs using the learnt method (SheafLap). In Table 5.3, we observe that, except for one case, the unnormalised sheaf Laplacian proves to be a better candidate. The normalisation Laplacian matrix can reduce the number of sign possibilities, as discussed in Section 4.4. Our results suggest that, in the case of the sheaf Laplacian, the benefit of normalisation to alleviate the sign ambiguity problem may not outweigh the expressive power of the unnormalised Laplacian in effectively differentiating nodes.

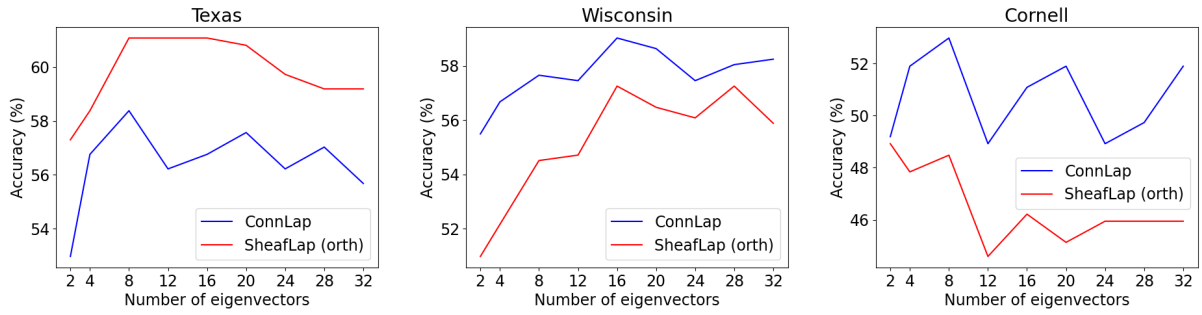


Figure 5.1: Accuracy (%) against number of eigenvectors used as PEs for ConnLap and SheafLap with an orthogonal matrix.

Number of eigenvectors Figure 5.1 presents our evaluation of the number of eigenvectors used as PEs. We can see that using more eigenvectors improves the performance until a turning point is reached, beyond which the accuracy deteriorates. This finding aligns with our claim regarding the sign ambiguity discussed in Section 4.4, that utilising more eigenvectors results in more sign possibilities and exacerbates the sign ambiguity problem. However, it is also crucial for nodes to have an adequate number of eigenvectors to obtain sufficient positional information. For example, in Wisconsin, 16 eigenvectors are needed to achieve the optimal performance for both ConnLap and SheafLap. Therefore, the choice of the number of eigenvectors is critical. Furthermore, we note that this trend is not observed in the Cornell dataset, suggesting that Laplacian-based PEs may not be

well suited for this graph. This finding helps explain the poor performance of SheafLap in Table 5.2.

5.1.3.1 Scalability

	Film	**Squirrel**	**Chameleon**	**Citeseer**	**Pubmed**	**Cora**
#Nodes	7,600	5,201	2,277	3,327	18,717	2,708
#Edges	26,752	198,493	31,421	4,676	44,327	5,278
No PE	**25.20±0.69**	46.62±3.62	63.97±3.10	72.34±1.41	**86.43±0.35**	84.71±1.23
GraphLap	25.13±0.99	47.56±3.03	64.28±3.00	73.83±2.07	86.43±0.36	85.05±1.47
ConnLap	**26.53±0.86**	**47.92±3.53**	**65.57±2.52**	**73.88±1.84**	**86.49±0.42**	**85.13±1.34**
SheafLap	23.80±1.10	**51.11±2.95**	**65.20±3.10**	**74.35±1.64**	85.84±0.65	**85.88±1.26**

Table 5.4: Mean±std accuracy for node-level tasks with larger graphs across 10 folds using different types of positional encodings. The best and the second best results are highlighted in **red** and **blue**, respectively.

We then evaluate sheaf-based PEs on other node-level tasks to test their scalability to larger sizes and adaptability to different densities. Since the graphs are much larger than those in Table 5.2, the computational costs are much higher, especially for sheaf learning. We choose to use SheafLap with an orthogonal matrix for the experiments here. This is because orthogonal sheaf balances the trade-off between generality and efficiency. Furthermore, it is the same type of sheaf used in ConnLap, providing a fairer comparison.

In Table 5.4, we observe that sheaf-based PEs (ConnLap, SheafLap) continue to outperform GraphLap, maintaining the top two best results in most datasets. This further indicates the effectiveness of our sheaf-based PEs, even on much larger datasets. Furthermore, these graphs cover different densities. While sparse graphs suffer more from locality, dense graphs may lead to similar neighbourhoods. Both sheaf-based PEs cope well with sparse graphs (such as Citeseer) and dense graphs (such as Squirrel). Within sheaf-based PEs, we can see SheafLap obtains higher gains in those where it wins. In these larger graphs, SheafLap is more likely to receive sufficient data to be properly trained, which may explain its outstanding performance. We also note that ConnLap remains more stable than SheafLap, due to its numerical stability by avoiding gradient-based learning.

5.2 Graph-level tasks

Graph-level tasks require mapping graphs to proper graph embeddings, which should be different for non-isomorphic graphs and identical for isomorphic ones. A graph embedding is usually computed via aggregating all node embeddings through operations such as sum or mean. Hence, PEs should capture appropriate positions of the nodes in the whole graph. This also requires higher-level coordination of these PEs compared to node-level tasks.

Datasets We use two sets of real-world molecular graphs (ZINC [71] and OGBG-MOLTOX21 [72]) for graph-level predictions. The ZINC dataset is a graph regression task to predict the constrained solubility (logP) value of each molecule. The node and edge features are provided as scalars, representing the atom or bond types. The OGBG-MOLTOX21 dataset requires classifying the molecules to a binary label, where the node features are 9-dimensional vectors. Appendix B provides a more detailed description for each dataset.

Setup For the ZINC dataset, we use the predefined 10K/1K/1K splits for training, validation, and testing, respectively. For the OGBG-MOLTOX21 dataset, we adopt the default splits by Open Benchmark Dataset [72] based on scaffolding splitting [73]. The hyperparameter setup follows [47], where GatedGCN, PNA, and SAN are used as base models. We select the 8 smallest eigenvectors as PEs, and a dimension $d = 3$ for the stalks in the sheaf. The reported results are the mean and standard deviation of metrics across 4 runs using different random seeds.

Evaluation metrics The two datasets use different evaluation metrics. ZINC is a graph regression task, which uses the Mean Absolute Error (MAE) to quantify the magnitude of errors. Therefore, the objective is to minimise the MAE. Conversely, OGBG-MOLTOX21 is a graph classification task, where Area Under the Curve (AUC) is used to measure the probability of predicting a positive example over a negative example. In this task, a higher AUC is desired.

5.2.1 Pre-computed connection Laplacian as PEs

For graph-level tasks, each dataset contains multiple graphs, posing a challenge for sheaf learning. During training, these graphs are fed into the model in batches. Due to the size differences in batched graphs, they are processed as a single large graph with disconnected smaller subgraphs. This requires us to "unbatch" the graphs to compute the learnt sheaf Laplacian for each graph during training, incurring an extraordinary cost. In contrast, the advantage of the connection Laplacian avoids the multi-graph problem by pre-computation before batching. Due to this constraint, we focus on ConnLap to test sheaf-based PEs for graph-level tasks. We leave the efficient treatment of multi-graphs in sheaf learning for future work.

In Table 5.5, we observe that not using PE (No PE) again results in the worst performance in both datasets, demonstrating the effectiveness of Laplacian-based PEs. Furthermore, ConnLap achieves the best result (77.9) compared to GraphLap (77.4) in MOLTOX21. This shows that the sheaf Laplacian, by accounting for node data in addition to the graph structure, can provide more informative PEs also in the global scope. However, in ZINC, we also note that ConnLap only achieves a marginal improvement (0.249) over No PE (0.251), whereas GraphLap has an outstanding performance (0.202). The possible reason

Base Model		ZINC TestMAE (↓)	ZINC+LSPE TestMAE (↓)	MOLTOX21 TestAUC (↑)
GatedGCN	No PE	0.251±0.009	N.A.	77.2±0.6
	GraphLap	**0.202±0.006**	0.196±0.008	77.4±0.7
	ConnLap	0.249±0.005	**0.193±0.014**	**77.9±0.2**

Table 5.5: Mean±std MAE (lower is better) for ZINC and mean±std AUC (%, higher is better) for OGBG-MOLTOX21 across 4 random seeds using different types of positional encodings. The best result is highlighted in **red**.

for this is that the node features in ZINC are merely scalar values, indicating the index of the atom type. Such sparse information is inadequate for constructing the connection Laplacian through the manifold method. To tackle this problem, we take the one-hot encodings of these scalars to obtain feature vectors. However, such one-hot embeddings may not be meaningful enough to fulfil the manifold assumption. On the other hand, node features in MOLTOX21 have 9 dimensions and are not blocked by this problem.

5.2.2 Learnable positional encodings

We allow the pre-computed ConnLap to evolve with training as explained in Section 4.2.4 to address the issue when the manifold assumption does not hold. We also compare GraphLap augmented with such a learnable mechanism for a fair comparison. The results are shown in the "ZINC+LSPE" column in Table 5.5. While GraphLap benefits slightly from such change (0.202 to 0.196), ConnLap gains significantly from learning (0.249 to 0.193). This observation aligns with our hypothesis. While the scalar node features are not rich enough to construct ConnLap during the preprocessing time, the sheaf structure sets a good starting point to let ConnLap evolve. Furthermore, as node representations become more meaningful across layers, the connection Laplacian can encode better global positions from the improved node features.

5.2.3 Different architectures

	GatedGCN	PNA	SAN
No PE	77.2±0.6	**75.5±0.8**	74.4±0.7
GraphLap	77.4±0.7	75.2±1.3	73.6±0.3
ConnLap	**77.9±0.2**	75.3±0.4	**74.5±0.4**

Table 5.6: Mean±std AUC for MOLTOX21 across 4 random seeds using different base models. The best result is highlighted in **red**.

Our sheaf-based PEs are model-agnostic. PEs are also essential to complement Graph Transformers with the input graph structure. In Table 5.6, we compare GraphLap and ConnLap across two GNN models (GatedGCN and PNA) and one Graph Transformer model (SAN). The model choices follow previous works on the graph Laplacian PEs [47,

45]. As discussed above, ConnLap has the best performance with GatedGCN (77.9) compared to GraphLap (77.4) and No PE (77.2). However, both PEs (75.2 and 75.3) do not match the vanilla PNA architecture. This suggests that Laplacian-based PEs may not be suitable for PNA, which increases the expressivity by combining multiple aggregators. One possible reason is that PNA does not follow the WL-test hierarchy [7, 21], so global structural knowledge from the graph may not be adequate to improve its expressivity. However, we note that ConnLap still outperforms GraphLap in this case and is more stable with an std of 0.4 versus 1.3. Moving on to SAN, which is a Graph Transformer, we notice that although GraphLap performs poorly (73.6) compared to No PE (74.4), Connlap brings a slight improvement (74.5) and higher stability as well (0.4). Overall, we can still observe the superior quality of ConnLap compared to GraphLap in terms of accuracy and stability.

5.2.4 Comparison with relative positional encodings

	GatedGCN	**PNA**	**SAN**
No PE	77.2±0.6	75.5±0.8	74.4±0.7
RWPE	77.5±0.3	**76.1±0.7**	74.4±0.8
ConnLap	**77.9±0.2**	75.3±0.4	**74.5±0.4**

Table 5.7: Mean±std AUC for MOLTOX21 dataset across 4 random seeds comparing ConnLap with a relative positional encoding based on random walks (RWPE). The best result is highlighted in **red**.

In Table 5.7, we compare ConnLap, being a global PE, with a relative PE based on random walks (RWPE) [47]. We can see ConnLap maintains the best performance with GatedGCN (77.9 vs 77.5) and SAN (74.5 vs 74.4), except in the case of PNA (75.3 vs 76.1), in comparison with RWPE. We discussed in the previous section that both global PEs using the Laplacian matrix tend to perform poorly with PNA. Here, we notice RWPE, as a relative PE, is more appropriate for this architecture. This supports our claim that PNA may not benefit from global structural knowledge, but relative distance information can facilitate its multiple aggregator scheme. Finally, we note that sheaf-based PEs can be transformed into relative PEs by taking the gradients of eigenvectors. We believe this leaves ample future avenues to explore.

5.3 Qualitative Evaluation

5.3.1 Synthetic graphs

We visualise the PEs generated from GraphLap and ConnLap by creating two synthetic graphs representing the chemical compounds decalin and bicyclopentyl. This is a well-known pair of graphs that are non-isomorphic but hard to distinguish by GNNs [9]. As

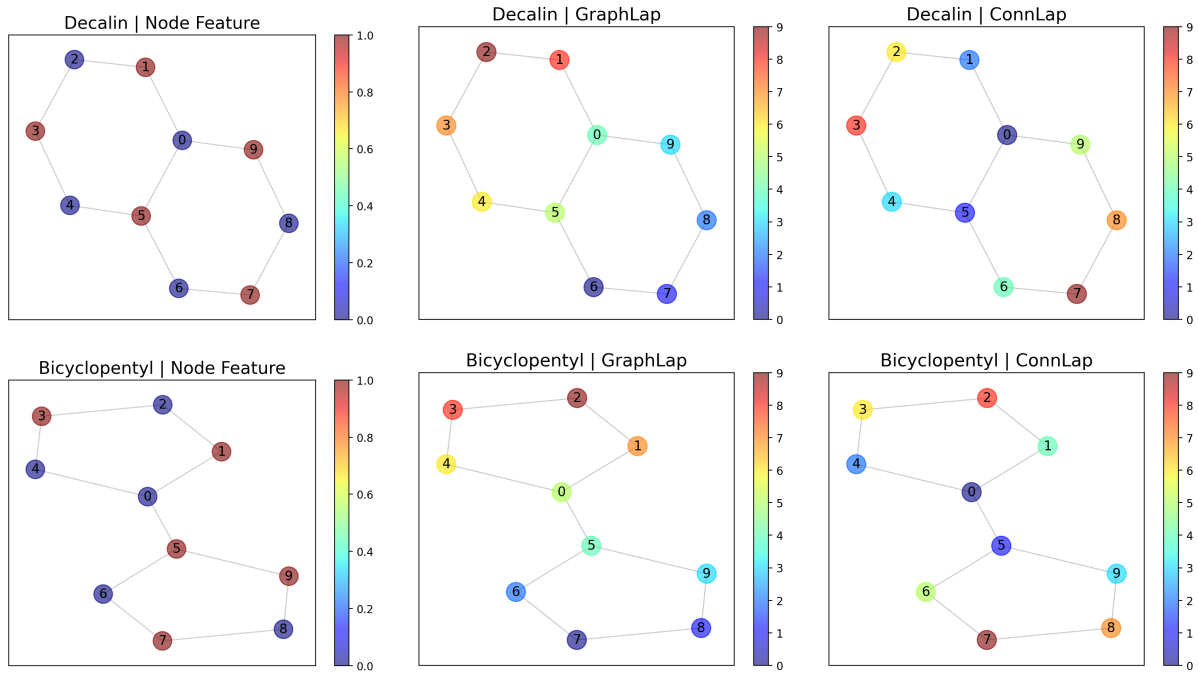


Figure 5.2: Visualisation of PEs created from GraphLap and ConnLap on two 10-node graphs with decalin and bicyclopentyl structures [9]. These graphs are heterophilic, where nodes connected may have different node features.

shown in Figure 5.2, we populate the node features with alternating vectors of 0s and 1s to make them heterophilic. The second and third columns display the visualisation of PEs from GraphLap and ConnLap, respectively, where similar colours indicate similar PEs. We see that GraphLap tends to generate a smoothly transiting colour scheme among neighbouring nodes. In contrast, ConnLap can generate dissimilar PEs when node features differ. For example, in decalin, GraphLap assigns either warm (red) or cold (blue) colours in each ring, while ConnLap assigns mixed colours. This demonstrates that ConnLap can effectively account for both structural and semantic information from the graph, generating a more expressive positional embedding space to facilitate node differentiation.

5.3.2 OGBG-MOLTOX21

We also take two real-world graphs randomly from the OGBG-MOLTOX21 dataset (index 295 and 399) to visualise the PEs generated by GraphLap and ConnLap. Similar observation can be made in Figure 5.3, where ConnLap assigns dissimilar colours to nodes by incorporating their data differences, in contrast to the smooth colour transition seen with GraphLap. This finding aligns with the better performance of ConnLap as shown in Table 5.5, suggesting that these PEs initialised by ConnLap can effectively improve the learning of graph embedding when aggregated from all nodes.

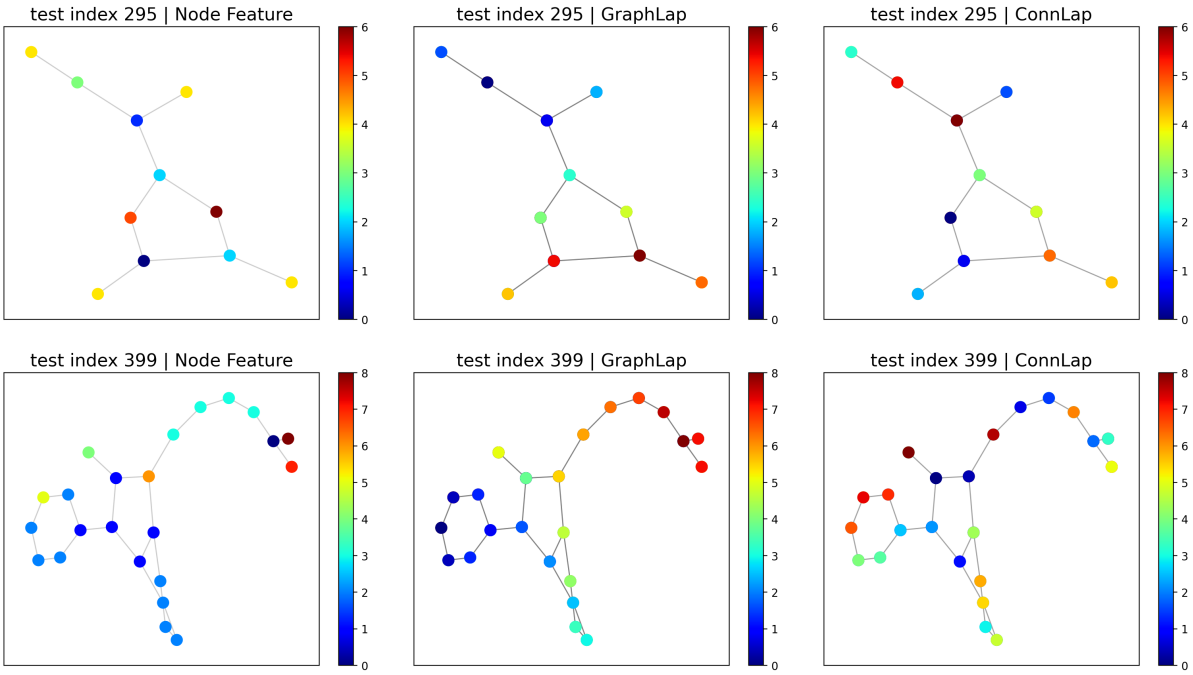


Figure 5.3: Visualisation of PEs created from GraphLap and ConnLap on 2 random graphs from OGBG-MOLTOX21 dataset.

5.4 Complexity Evaluation

	Texas		**Citeseer**		**Squirrel**	
	pre-compute	per epoch	pre-compute	per epoch	pre-compute	per epoch
GraphLap	49.67	6.71	1886.58	4.47	9654.07	41.87
ConnLap	608.32	7.79	8562.46	3.48	72148.34	33.98
SheafLap	0	19.03	0	3139.45	0	31503.36

Table 5.8: Mean execution time in seconds (s) with different types of PEs on node-level tasks, sorted in increasing size of graph. In each dataset, "pre-compute" displays the total time cost to construct GraphLap/ConnLap during preprocessing time, and "per epoch" is the averaged time cost per epoch.

We compare the mean execution time for each type of PE on different datasets, trained on NVIDIA Tesla T4 GPU. In Table 5.8 and Table 5.9, we observe that the pre-computation time costs for ConnLap exceed those of GraphLap (around $\times 10$), due to the higher complexity for constructing the sheaf Laplacian than the graph Laplacian. However, this is a one-off cost per dataset, and the time per epoch remains similar for both PEs. In Table 5.8, SheafLap induces a much higher cost per epoch (around $\times 1000$), which is significantly costlier considering that each training procedure typically requires hundreds of epochs. Although SheafLap does not require pre-computation, its high computational cost per epoch makes it less scalable than GraphLap and ConnLap.

	ZINC		ZINC+LSPE	MOLTOX21	
	pre-compute	per epoch	per epoch	pre-compute	per epoch
GraphLap	30.63	15.53	22.81	23.45	4.98
ConnLap	372.72	15.47	23.24	196.52	5.01

Table 5.9: Execution time in seconds (s) with different types of PEs on graph-level tasks, where ZINC+LSPE indicates allowing PEs to evolve during training.

5.5 Summary

As a summary, our main findings are:

- PEs constructed from the Laplacian matrix help improve the expressivity of vanilla GNNs and Graph Transformers in both node-level and graph-level tasks.
- Sheaf-based PEs (ConnLap, SheafLap) are more expressive than GraphLap with the ability to additionally encode the relationships between the node data. This claim is further supported by qualitative analysis.
- It is critical to choose an appropriate number of eigenvectors to maximise the effectiveness of sheaf-based PEs. Furthermore, an unnormalised sheaf Laplacian is more suitable to construct PEs.
- ConnLap is more stable and computationally efficient than SheafLap due to the advantage that it can be pre-computed. However, its quality is limited by whether the input data fulfils the manifold assumption. This problem can be alleviated by adding a learning mechanism to allow it to evolve.
- Learnt SheafLap is more general than ConnLap by avoiding the manifold assumption. It can achieve outstanding results in many tasks, but it is harder to train and may suffer from numerical instability. Its engineering bottleneck in multi-graph settings also sets a constraint in graph-level tasks.

Chapter 6

Conclusion

6.1 Accomplishments

Positional encodings serve as effective means of enhancing the expressivity of GNNs and are essential for complementing Graph Transformers with valuable graph structural information. In this project, we propose a novel design of PEs using sheaf theory. Our experiments demonstrate that sheaf-based PEs outperform existing methods based on the graph Laplacian. We show the importance of semantic information to capture the complex relationships between the node data, resulting in a more expressive embedding space to encode the algebraic and topological structure of the graph. Furthermore, we design two variants of sheaf-based PEs using pre-computed and learnt methods, exploring their trade-offs between computational complexity and expressive power.

Empirically, in addition to the graph-level tasks, which have been the focus of existing literature, we also extensively assess PEs on node-level tasks spanning various graph sizes, densities, and homophily levels. Our results indicate that sheaf-based PEs are particularly good at node differentiation, exposing the limitation of the graph Laplacian where the node data is not taken into account.

Furthermore, our work contributes to the application of sheaf theory in graph learning. While previous works have concentrated on modelling the convolution operator in GNNs as a sheaf diffusion process, we take a different approach by encoding the graph structure using a sheaf. Our work further highlights the significance of sheaf theory in GNNs, prompting a shift in the current focus of spectral GNNs from the graph Laplacian to the more generalised sheaf Laplacian.

Finally, our work showcases the value of drawing inspiration from geometry and topology to develop novel techniques for geometric deep learning, which encompasses the study of GNNs. These mathematical tools have been extensively studied in theory, inviting ample opportunities to explore their implications in learning geometric data.

6.2 Future work

We now point out possible future directions that were not explored due to the time and computation resource constraints of this project.

SheafLap in multi-graph settings As discussed in Section 5.2.1, SheafLap was not evaluated for graph-level tasks due to its bottleneck in unbatching the graphs. More engineering effort can be invested in handling the graph batches during sheaf construction and reducing its computational cost.

Sheaf-based PE loss The learnable mechanism that we allow ConnLap to evolve does not use any PE-specific loss. In [47], they propose to use a Laplacian eigenvector loss [11] to align the PE with the graph topology. Therefore, designing a sheaf-based PE loss to enforce ConnLap to follow the sheaf structure is also worth exploring.

SignNet integration We dealt with the sign ambiguity problem by randomly flipping the sign of eigenvectors during training. A recent proposal, SignNet [45], learns the sign invariance explicitly to address this issue. We can integrate SignNet with the sheaf-based PEs to see if it enhances the performance.

Relative PEs and structural encodings extension Finally, we can extend our sheaf-based global PEs to relative PEs by taking the gradients of the eigenvectors, and then compare with other relative PEs such as RWPE [47]. Similarly, sheaf-based PEs can also be transformed into structural encodings by appending the eigenvalues of the sheaf Laplacian to the graph embedding. The computation of eigenvalues is numerically stable during backpropagation, which may offer improvement in graph-level tasks.

Bibliography

[1] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907.

[2] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. "Neural Message Passing for Quantum Chemistry". In: *ICML*. 2017.

[3] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. "Graph Attention Networks". In: *International Conference on Learning Representations*. 2018.

[4] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. "Protein Interface Prediction using Graph Convolutional Networks". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[5] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. *Graph Neural Networks for Social Recommendation*. 2019. arXiv: 1902.07243 [cs.IR].

[6] Kun Xu, Liwei Wang, Mo Yu, Yansong Feng, Yan Song, Zhiguo Wang, and Dong Yu. "Cross-lingual Knowledge Graph Alignment via Graph Matching Neural Network". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3156–3161. DOI: 10.18653/v1/P19-1304.

[7] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. *How Powerful are Graph Neural Networks?* 2018. DOI: 10.48550/ARXIV.1810.00826.

[8] Petar Veličković. "Message passing all the way up". In: *ICLR 2022 Workshop on Geometrical and Topological Representation Learning*. 2022.

[9] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. "Approximation Ratios of Graph Neural Networks for Combinatorial Problems". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *NIPS*. 2017.

[11] Mikhail Belkin and Partha Niyogi. "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation". In: *Neural Computation* 15.6 (2003), pp. 1373–1396. DOI: 10.1162/089976603321780317.

[12] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. *Benchmarking Graph Neural Networks*. 2020. DOI: 10.48550/ARXIV.2003.00982.

[13] Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. *Rethinking Graph Transformers with Spectral Attention*. 2021. arXiv: 2106.03893 [cs.LG].

[14] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[16] Xavier Bresson and Thomas Laurent. *Residual Gated Graph ConvNets*. 2018. arXiv: 1711.07553 [cs.LG].

[17] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. "Principal Neighbourhood Aggregation for Graph Nets". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 13260–13271.

[18] Michael Bronstein. *Expressive power of graph neural networks and the Weisfeiler-Lehman test*. 2020.

[19] Boris Weisfeiler and Andrei Lehman. "The reduction of a graph to canonical form and the algebra which appears therein". In: *NTI Series* 2(9):12-16 (1968).

[20] Laszlo Babai and Ludik Kucera. "Canonical labelling of graphs in linear average time". In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. 1979, pp. 39–46. DOI: 10.1109/SFCS.1979.8.

[21] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. *Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks*. 2018. DOI: 10.48550/ARXIV.1810.02244.

[22] J.-Y. Cai, M. Furer, and N. Immerman. "An optimal lower bound on the number of variables for graph identification". In: *30th Annual Symposium on Foundations of Computer Science*. 1989, pp. 612–617. DOI: 10.1109/SFCS.1989.63543.

[23] B. L. Douglas. *The Weisfeiler-Lehman Method and Graph Isomorphism Testing*. 2011. arXiv: 1101.5211 [math.CO].

[24] Sergei Evdokimov and Ilia Ponomarenko. "Isomorphism of Coloured Graphs with Slowly Increasing Multiplicity of Jordan Blocks". In: *Combinatorica* 19 (Mar. 1999), pp. 321–333. DOI: 10.1007/s004930050059.

[25] Jing Huang and Jie Yang. "Unignn: a unified framework for graph and hypergraph neural networks". In: *arXiv preprint arXiv:2105.00956* (2021).

[26] Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. *You are AllSet: A Multiset Function Framework for Hypergraph Neural Networks*. 2022. arXiv: 2106.13264 [cs.LG].

[27] Dobrik Georgiev, Marc Brockschmidt, and Miltiadis Allamanis. *HEAT: Hyperedge Attention Networks*. 2022. arXiv: 2201.12113 [cs.LG].

[28] Cristian Bodnar, Fabrizio Frasca, Yu Guang Wang, Nina Otter, Guido Montúfar, Pietro Liò, and Michael Bronstein. *Weisfeiler and Lehman Go Topological: Message Passing Simplicial Networks*. 2021. DOI: 10.48550/ARXIV.2103.03212.

[29] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yu Guang Wang, Pietro Liò, Guido Montúfar, and Michael Bronstein. *Weisfeiler and Lehman Go Cellular: CW Networks*. 2021. DOI: 10.48550/ARXIV.2106.12575.

[30] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. *From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness*. 2022. arXiv: 2110.03753 [cs.LG].

[31] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. *Equivariant Subgraph Aggregation Networks*. 2022. arXiv: 2110.02910 [cs.LG].

[32] Jiaxuan You, Rex Ying, and Jure Leskovec. *Position-aware Graph Neural Networks*. 2019. DOI: 10.48550/ARXIV.1906.04817.

[33] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. *Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning*. 2020. DOI: 10.48550/ARXIV.2009.00142.

[34] Vijay Prakash Dwivedi and Xavier Bresson. "A Generalization of Transformer Networks to Graphs". In: *CoRR* abs/2012.09699 (2020). arXiv: 2012.09699.

[35] Qimai Li, Zhichao Han, and Xiao-Ming Wu. "Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, 2018, pp. 3538–3545.

[36] Uri Alon and Eran Yahav. *On the Bottleneck of Graph Neural Networks and its Practical Implications*. 2020. DOI: 10.48550/ARXIV.2006.05205.

[37] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. *Do Transformers Really Perform Bad for Graph Representation?* 2021. arXiv: 2106.05234 [cs.LG].

[38] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. *GraphiT: Encoding Graph Structure in Transformers*. 2021. arXiv: 2106.05667 [cs.LG].

[39] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. *Recipe for a General, Powerful, Scalable Graph Transformer*. 2023. arXiv: 2205.12454 [cs.LG].

[40] Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. *Attending to Graph Transformers*. 2023. arXiv: 2302.04181 [cs.LG].

[41] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. *Deep Sets*. 2018. arXiv: 1703.06114 [cs.LG].

[42] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[43] Md Amirul Islam, Sen Jia, and Neil D. B. Bruce. *How Much Position Information Do Convolutional Neural Networks Encode?* 2020. arXiv: 2001.08248 [cs.CV].

[44] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. *Relational Pooling for Graph Representations*. 2019. DOI: 10.48550/ARXIV.1903.02541.

[45] Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. *Sign and Basis Invariant Networks for Spectral Graph Representation Learning*. 2022. DOI: 10.48550/ARXIV.2202.13013.

[46] Sohir Maskey, Ali Parviz, Maximilian Thiessen, Hannes Stärk, Ylli Sadikaj, and Haggai Maron. *Generalized Laplacian Positional Encoding for Graph Representation Learning*. 2022. arXiv: 2210.15956 [cs.LG].

[47] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. "Graph Neural Networks with Learnable Structural and Positional Representations". In: (2021). DOI: 10.48550/ARXIV.2110.07875.

[48] Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. *Equivariant and Stable Positional Encoding for More Powerful Graph Neural Networks*. 2022. DOI: 10.48550/ARXIV.2203.00199.

[49] Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. *Sign and Basis Invariant Networks for Spectral Graph Representation Learning*. 2022. DOI: 10.48550/ARXIV.2202.13013.

[50] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. *GraphiT: Encoding Graph Structure in Transformers*. 2021. DOI: 10.48550/ARXIV.2106.05667.

[51] Or Feldman, Amit Boyarski, Shai Feldman, Dani Kogan, Avi Mendelson, and Chaim Baskin. *Weisfeiler and Leman Go Infinite: Spectral and Combinatorial Pre-Colorings*. 2022. DOI: 10.48550/ARXIV.2201.13410.

[52] Justin Curry. *Sheaves, Cosheaves and Applications*. 2014. arXiv: 1303.3255 [math.AT].

[53] Amit Singer and Hau-tieng Wu. *Vector Diffusion Maps and the Connection Laplacian*. 2011. arXiv: 1102.0075 [math.ST].

[54] Cristian Bodnar, Francesco Di Giovanni, Benjamin Paul Chamberlain, Pietro Liò, and Michael M. Bronstein. *Neural Sheaf Diffusion: A Topological Perspective on Heterophily and Oversmoothing in GNNs*. 2022. DOI: 10.48550/ARXIV.2202.04579.

[55] Jakob Hansen and Thomas Gebhart. *Sheaf Neural Networks*. 2020. DOI: 10.48550/ARXIV.2012.06333.

[56] Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, Michael Bronstein, Petar Veličković, and Pietro Liò. *Sheaf Neural Networks with Connection Laplacians*. 2022. arXiv: 2206.08702 [cs.LG].

[57] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José M. F. Moura, and Pierre Vandergheynst. "Graph Signal Processing: Overview, Challenges, and Applications". In: *Proceedings of the IEEE* 106.5 (2018), pp. 808–828. DOI: 10.1109/JPROC.2018.2820126.

[58] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. *Spectral Networks and Locally Connected Networks on Graphs*. 2014. arXiv: 1312.6203 [cs.LG].

[59] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. "Spectral grouping using the Nystrom method". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.2 (2004), pp. 214–225. DOI: 10.1109/TPAMI.2004.1262185.

[60] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. *Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections*. 2017. arXiv: 1612.00188 [cs.LG].

[61] Muhammad F. Kasim and Sam M. Vinko. *ξ-torch: differentiable scientific computing library*. 2020. arXiv: 2010.01921 [cs.LG].

[62] Muhammad Firmansyah Kasim. *Derivatives of partial eigendecomposition of a real symmetric matrix for degenerate cases*. 2020. arXiv: 2011.04366 [math.NA].

[63] Peter Arbenz. *Lecture Notes on Solving Large Scale Eigenvalue Problems*. 2016.

[64] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. "Geom-GCN: Geometric Graph Convolutional Networks". In: *International Conference on Learning Representations*. 2020.

[65] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. "Collective Classification in Network Data". In: *AI Magazine* 29.3 (Sept. 2008), p. 93. DOI: 10.1609/aimag.v29i3.2157.

[66] Galileo Namata, Ben London, Lise Getoor, and Bert Huang. "Query-driven Active Surveying for Collective Classification". In: 2012.

[67] *CMU World Wide Knowledge Base (Web-¿KB) project*. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/.

[68] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. "Social Influence Analysis in Large-Scale Networks". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '09. Paris, France: Association for Computing Machinery, 2009, pp. 807–816. ISBN: 9781605584959. DOI: 10.1145/1557019.1557108.

[69] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. *Multi-scale Attributed Node Embedding*. 2021. arXiv: 1909.13021 [cs.LG].

[70] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. *Geom-GCN: Geometric Graph Convolutional Networks*. 2020. DOI: 10.48550/ARXIV.2002.05287.

[71] John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. "ZINC: A Free Tool to Discover Chemistry for Biology". In: *Journal of Chemical Information and Modeling* 52.7 (2012). PMID: 22587354, pp. 1757–1768. DOI: 10.1021/ci3001277. eprint: https://doi.org/10.1021/ci3001277.

[72] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: *Advances in Neural Information Processing Systems*.

Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 22118–22133.

[73] Zhenqin Wu, Bharath Ramsundar, Evan Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh Pappu, Karl Leswing, and Vijay Pande. "MoleculeNet: A Benchmark for Molecular Machine Learning". In: *Chemical Science* 9 (Mar. 2017). DOI: 10.1039/C7SC02664A.

Appendix A

Sheaf Neural Networks

We provide an in-depth description of Sheaf Neural Networks. Given $\mathbf{X}$ as the node feature matrix, the operation of GNNs can be understood from the perspective of a heat diffusion process:

$$\mathbf{X}_0 = \mathbf{X}, \quad \dot{\mathbf{X}}_t = -\Delta \mathbf{X}_t \tag{A.1}$$

GNNs use the discretised version of such a continuous diffusion process, which implicitly involves the graph Laplacian. Take Graph Convolutional Network [1] as an example, by performing Euler discretisation on Equation A.1, its update rule can be represented as:

$$\mathbf{X}_{l+1} = \sigma(\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{X}_l\mathbf{W}) = \sigma((\mathbf{I} - \Delta_0)\mathbf{X}_l\mathbf{W}) \tag{A.2}$$

where $\mathbf{A}$ is the adjacency matrix, $\mathbf{D}$ is the degree matrix, and Δ_0 is the normalised graph Laplacian. Furthermore, $\mathbf{W}$ is a weight matrix with σ as a non-linear activation function.

Sheaf Neural Networks The work by [55] generalises such graph diffusion operation (Equation A.2) from the graph Laplacian (Δ_0) to the sheaf Laplacian ($\Delta_{\mathcal{F}}$):

$$\mathbf{X}_{l+1} = \sigma\Big((\mathbf{I}_{nd} - \Delta_{\mathcal{F}})(\mathbf{I}_n \otimes \mathbf{W}_1)\mathbf{X}_l\mathbf{W}_2\Big) \tag{A.3}$$

where $\mathbf{W}_1$ is the left weight matrix, $\mathbf{W}_2$ is the right weight matrix, and $\otimes$ is the Kronecker product. When the sheaf is constant, i.e. $\Delta_{\mathcal{F}} = \Delta_0$, the left weight matrix $\mathbf{W}_1$ is a scalar and the diffusion process becomes Equation A.2.

Since the sheaf Laplacian is able to represent more complex relationships between the node data, they empirically showed that Sheaf Neural Networks are advantageous over node data that are non-constant, asymmetric, and varying in dimension. This setting is commonly understood as heterophilic, where nearby nodes do not share similar node

features. The superior expressivity of sheaf diffusion is formally proved in [54] using a hierarchy of more generalised sheaves within the infinite time limit.

Neural sheaf diffusion The method proposed in [55] requires full knowledge of a graph to construct the sheaf Laplacian, making it difficult to generalise to unseen graphs. A learning-based approach was proposed in [54], where restriction maps are set as a learnable parametric function, allowing the sheaf to be learnt from end-to-end. Furthermore, they propose neural sheaf diffusion that utilises a sheaf with higher-dimensional stalks and a residual parameterisation:

$$\mathbf{X}_{l+1} = \mathbf{X}_l - \sigma\Big(\Delta_{\mathcal{F}(l)}(\mathbf{I} \otimes \mathbf{W}_1^l)\mathbf{X}_l\mathbf{W}_2^l\Big) \tag{A.4}$$

where the sheaf $\Delta_{\mathcal{F}(l)}$ and weight matrices $\mathbf{W}_1^l, \mathbf{W}_2^l$ evolve at each layer l. This mechanism ensures that as node representations become more meaningful at each layer, the underlying sheaf also gets updated to encode the new node data. Empirically, the expressive power of sheaves is demonstrated in a set of increasingly heterophilic graph datasets.

Connection Laplacian However, the computational cost of sheaf learning can be expensive when the graph is large. The numerical instability associated with gradient-based learning for sheaf also poses a challenge. In [56], they propose a way to pre-compute the sheaf Laplacian with orthogonal maps from the input data in a deterministic manner. The method assumes that the data comes from a manifold. Then, the sheaf Laplacian can be computed via optimal alignment of neighbouring tangent spaces through orthogonal transformations. Due to the analogy to parallel transport (connection), such type of sheaf Laplacian is also called the connection Laplacian. The empirical results showed that the connection Laplacian maintains the superior expressivity from neural sheaf diffusion, while greatly reduces computational costs.

Appendix B

More on datasets

B.1 Node-level tasks

Citation networks (Cora, Citeseer, Pubmed) In these citation networks [65, 66], nodes are academic papers with edges showing their citation relationships. The node features are the bag-of-words representation of the papers. The goal is to predict the academic topic of each paper.

WebKB (Cornell, Texas, Wisconsin) These are webpage datasets [67] collected from computer science departments in various universities. The nodes are webpages, while the edges are hyperlinks between them. Similarly, the node features are the bag-of-words representation, with the goal to classify the category of the webpage.

Actor co-occurrence network (Film) This dataset [68] is a subgraph of the film-director-actor-writer network, with nodes representing actors and edges denoting co-occurrence on the same Wikipedia page. The node features consist of keywords in the Wikipedia page, while the node label corresponds to the actor's Wikipedia category.

Wikipedia network (Chameleon, Squirrel) The two Wikipedia networks [69] consist of pages related with chameleon or squirrel. The nodes are Wikipedia pages, and the edges are mutual inks between them. Node features contain information nouns from the page. The task requires to classify the pages into five categories based on their average monthly traffic.

B.2 Graph-level tasks

ZINC The ZINC dataset contains 12K molecular graphs representing commercially available chemical compounds. The molecular graphs vary in size, from 9 to 37 nodes. Each node or edge is given a number as the feature, representing its atom or bond type.

In total, there are 28 atom types and 3 bond types. This graph regression task requires the model to predict a constrained solubility (logP) value of the compound.

OGBG-MOLTOX21 This dataset is a smaller dataset taken from MoleculeNet [73]. The nodes are atoms with 9-dimensional vector features, encoding information such as atomic number, chirality, and formal charge. The edges represent chemical bonds. It is a graph-level classification task with binary labels.