UNIVERSITY OF
CAMBRIDGE
Department of Computer
Science and Technology

HyperSheaf: a sheaf hypergraph library for heterogeneous data

Luke Braithwaite

Peterhouse

June 2024

Submitted in partial fulfillment of the requirements for the
Master of Philosophy in Advanced Computer Science

Total page count: 72

Main chapters (excluding front-matter, references and appendix): 50 pages (pp 9–58)

Main chapters word count: 11 105

Methodology used to generate that word count:

```
$ texcount report.tex -inc -1 -sum
```

Declaration

I, Luke Braithwaite of Peterhouse, being a candidate for the Master of Philosophy in Advanced Computer Science, hereby declare that this project report and the work described in it are my own work, unaided except as may be specified below, and that the project report does not contain material that has already been used to any substantial extent for a comparable purpose. In preparation of this project report I did not use text from AI-assisted platforms generating natural language answers to user queries, including but not limited to ChatGPT. I am content for my project report to be made available to the students and staff of the University.

The Sheaf Graph Neural Network code is adapted from Bodnar et al. [6]. The Sheaf Hypergraph Neural Network implementation is adapted from Duta et al. [14]. Figure 2.2 is based on a figure in the lecture slides of *Lecture 4 Graph Neural Networks* from *L65 Geometric Deep Learning*. The hypergraph image in Figure 2.5 is from Wikipedia and licensed under CC BY-SA 3.0.

Signed: Luke Braithwaite

Date: 17th August 2024

Abstract

Relational data is found across the many fields and extends beyond the pairwise interactions modelled by graph-based machine learning methods. This has led to an explosion of interest in Topological Deep Learning to process these more complex higher-order interactions. One such domain is hypergraphs, which extend graphs by connecting an arbitrary set of nodes, and have been extensively studied with many hypergraph-based methods developed. These approaches struggle to process heterogeneous hypergraphs that involve many different types of entities and relationships.

Inspired by recent work on cellular sheaves, this project explores how cellular sheaves can be attached to a hypergraph to add additional structure to mediate heterogeneous message passing. I demonstrate that adjusting sheaf learners to account for type information is beneficial with an increase in performance of up to 2 %. These **heterogeneous sheaf learners** are tested on several heterogeneous graph and hypergraph benchmarks, achieving competitive or state-of-the-art performance across each dataset.

To generalise existing sheaf-based architectures, I propose the **Sheaf-MPNN** and **SheafAllSet** methodologies that generalise sheaf message passing to graphs and hypergraphs, respectively. These approaches are used to integrate DeepSet and SetTransformer with sheaf message passing to create a series of neural architectures that are more flexible and expressive than existing sheaf-based architectures. They are **SheafDeepSet** and **SheafSetTransformer** for graphs with **SheafDeepAllSets** and **SheafAllSetTransformers** for hypergraphs. Finally, I develop the **HyperSheaf** library for sheaf-based multi-modal hypergraph neural networks which provides a series of implementations for all of the architectures proposed in this project.

The source code for this project is available at: https://github.com/AspieCoder1/mphil-acs-repo

Acknowledgements

I would like to thank my supervisor **Prof. Pietro Liò** and my co-supervisor **Iulia Duta** for their guidance and support throughout this project, without which this project would not have been possible.

This work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service (www.csd3.cam.ac.uk), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/T022159/1), and DiRAC funding from the Science and Technology Facilities Council (www.dirac.ac.uk).

Contents

1 Introduction **9**

1.1 Project aims . . . 11
1.2 Achievements and novel contributions . . . 11
1.3 Project structure . . . 12

2 Background **13**

2.1 Homogenous and heterogenous graphs . . . 13
2.2 Graph Neural Networks . . . 14
2.2.1 GNNs as modified a heat diffusion process . . . 15
2.2.2 Oversmoothing . . . 15
2.3 Cellular sheaves . . . 15
2.3.1 Sheaves prevent oversmoothing? . . . 17
2.4 SheafGNNs . . . 18
2.4.1 Sheaf Neural Networks . . . 18
2.4.2 Neural Sheaf Diffusion . . . 19
2.4.3 Sheaf Attention Networks . . . 20
2.5 Hypergraphs . . . 20
2.6 AllSet: message passing for hypergraphs . . . 21
2.7 Sheaf Hypergraph Neural Networks . . . 23
2.7.1 Sheaf hypergraph Laplacians . . . 23
2.7.2 Sheaf Hypergraph Networks . . . 24
2.8 Topological deep learning . . . 25

3 Related work **26**

3.1 Homogeneous GNNs . . . 26
3.2 Heterogeneous GNN architectures . . . 27

4 Heterogeneous sheaf learners **29**

4.1 Sheaves for heterogeneous data . . . 29
4.2 Including type information . . . 29
4.2.1 Embedding type information . . . 30
4.2.2 Ensemble-based sheaf learners . . . 32

4.3 Lifting to hypergraphs . . . 32
4.4 Computational complexity . . . 33

5 Experimental results **36**
5.1 Heterogeneous node classification . . . 37
5.1.1 Results . . . 37
5.2 Heterogeneous link prediction . . . 40
5.2.1 Results . . . 41
5.3 Heterogeneous drug-target interaction prediction . . . 43
5.3.1 Results . . . 44
5.4 Discussion . . . 46

6 A general sheaf message passing framework **48**
6.1 Sheaf Message Passing Neural Networks . . . 48
6.2 SheafAllSet: sheaf message passing for hypergraphs . . . 49
6.3 Learning Sheaf-MPNN and SheafAllSet layers . . . 50
6.4 Attaching sheaves to MPNNs . . . 52
6.5 HyperSheaf: a SheafHNN library for heterogeneous data . . . 53
6.6 Discussion . . . 55

7 Summary and conclusions **56**
7.1 Project summary . . . 56
7.2 Code availability and software engineering . . . 56
7.3 Future work . . . 58
7.3.1 Extending the HyperSheaf library . . . 58
7.3.2 Into domains topological . . . 58

References **59**

A Chapter 5 Supplementary Material **67**
A.1 Dataset information . . . 67
A.1.1 Heterogeneous node classification . . . 67
A.1.2 Heterogeneous link prediction . . . 67
A.1.3 Heterogeneous DTI prediction . . . 67
A.2 Restriction map implementation . . . 68

B Chapter 6 supplementary material **69**
B.1 Proofs . . . 69

Acronyms

GNN Graph Neural Network.

HNN Hypergraph Neural Network.

MLP Multi-Layer Perceptron.

MPNN Message Passing Neural Network.

NSD Neural Sheaf Diffusion.

Sheaf-MPNN Sheaf Message Passing Neural Network.

SheafGNN Sheaf Graph Neural Network.

SheafHNN Sheaf Hypergraph Neural Network.

TDL Topological Deep Learning.

TNN Topological Neural Network.

List of Figures

1.1 Illustration of four overlapping image sensors observing an image 10

2.1 Homogeneous and heterogeneous graphs. 13
2.2 The Graph Neural Network 14
2.3 Oversmoothing on a heterogeneous graph. 16
2.4 Illustration of a sheaf attached to a set 16
2.5 Example hypergraph and its representations 21
2.6 Illustration of the AllSets framework 22
2.7 A taxonomy of topological domains 25

3.1 Meta-path ensemble heterogeneous GNNs 27

4.1 Proposed heterogeneous sheaf learners 30

5.1 Linear probing of the learnt restriction maps based on edge types 40

6.1 General sheaf message passing 49
6.2 Illustration of SheafAllSet 50

7.1 Overview of work completed for the project 57

List of Tables

4.1 Computational complexities of heterogeneous sheaf learners 34

5.1 Dataset statistics for node classification datasets 37
5.2 Performance on heterogeneous node classification 38
5.3 Restriction map type ablation study 39
5.4 Computational overhead of sheaf learners 40
5.5 Dataset statistics for link prediction datasets 41
5.6 Performance on heterogeneous link prediction benchmarks 42
5.7 Restriction map ablation study for heterogeneous link prediction 43
5.8 Dataset statistics for DTI datasets 44
5.9 Performance on heterogeneous DTI prediction benchmarks 45
5.10 Restriction map ablation study for heterogeneous DTI prediction 45
5.11 Impact of hyperedge feature types on model performance 46

Chapter 1

Introduction

The prevalence of relational data in fields such as medicine [65], drug discovery [38], recommender systems [75] and computer vision tasks [41] has led to substantial interest in graph-based methods. This has led to the development of *Geometric Deep Learning* [7], which attempts to provide a principled approach to machine learning by providing a framework to exploit the geometry and structure inherent in data. However, higher-order interactions are prevalent in fields such as medical imaging [20, 21], neuroscience [25], social networks [1] and chemistry [39]. This has led to the introduction of hypergraph-based methods and, recently, the field of Topological Deep Learning (TDL) [26, 53]. Hypergraphs extend graphs beyond pairwise interactions by allowing arbitrary sets of nodes to be connected. This project explores how sheaf-based neural architectures can process heterogeneous hypergraphs, a subset of TDL that extends graphs to model higher-order interactions involving many different types of nodes and hyperedges.

Citation graphs are commonly modelled as heterogeneous graphs. For example, the DBLP citation graph for computer science contains information about a paper, its authors, keywords, and where it was published. Cui et al. [11] demonstrates that electronic health records can be treated as a heterogeneous hypergraph, where the nodes model medical observations such as prescriptions, test results or imaging data and the hyperedges connect all the nodes related to a visit and contain general clinical notes and a summary of the visit.

Lv et al. [46] surveyed different approaches of processing heterogeneous graph data. Standard Graph Neural Networks (GNNs) struggle to process heterogeneous data due to *oversmoothing*. This is where neighbouring nodes have increasingly similar representations as the message passing process continues, particularly with deeper GNNs with multiple message passing layers. In the case of heterogeneous graphs and hypergraphs, this causes all of the type-specific features and information to be lost during the message passing process, greatly reducing the model's performance. Recent literature [3, 4, 6, 13] has demonstrated that attaching a *cellular sheaf* [12, 63] to a graph helps mitig-

ate oversmoothing. However, using sheaf-based methods to model heterogeneous data remained unexplored.

Rosiak [58] motivates sheaves as a construction in which a series of local data assignments (called sections) may be combined to form a coherent global representation or data assignment (called the global section). Consider a set of four overlapping sensors that observe an image (Figure 1.1), each observing some local data, which are then 'glued' together to create global data, in this case, the image. This creates a construction that takes a series of compatible local data assignments and then 'glues' them together piece by piece and constrained by one another to build a global representation of the image. This results in a *sheaf*, where the individual images form the sections and the global section is the combined images formed by combining the individual images to create a coherent final result. The example is somewhat contrived but illustrates the underlying idea.

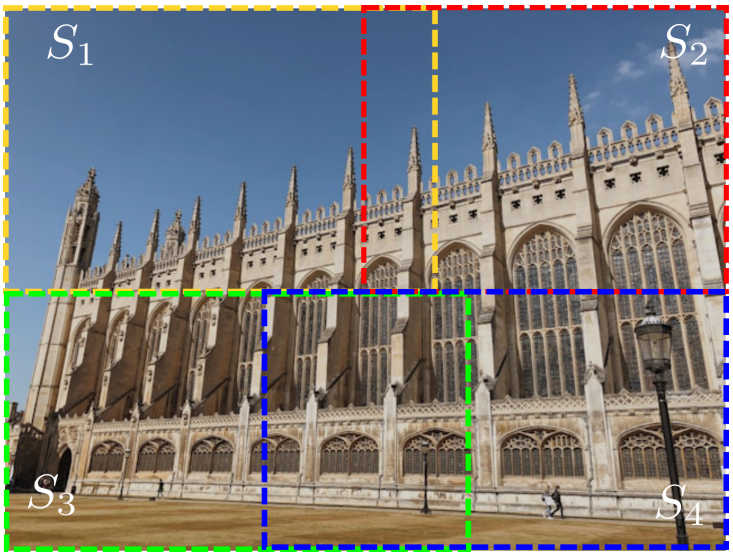


Figure 1.1: **Illustration of four overlapping image sensors observing an image.** Photo by Markus Leo on Unsplash.

Existing heterogeneous GNNs [36, 61, 73, 80] attempt to account for the heterogeneity in the model architecture. However, the sheaf assigns a series of local data to each node and edge in the graph, which are then connected together to create a global representation. This means that each edge and node can have completely different local meanings if they can be connected to create a meaningful global representation. Put differently, the sheaf implicitly models the heterogeneity in the data. This means that the model architecture is no longer required to account for or model the data's heterogeneity. The sheaf handles it for us.

1.1 Project aims

This project explores how equipping heterogeneous graphs and hypergraphs with cellular sheaves may improve performance on downstream tasks. I aim to answer the following research questions:

- Does sheaf-based processing improve performance on heterogeneous datasets?
- What is the best way to incorporate heterogeneous information into the sheaf framework?
- Is there a general framework that can describe sheaf graph and hypergraph neural networks and be consistent with existing literature?

Additionally, I aim to implement all of the proposed architectures and methods into a library, HyperSheaf, to allow other researchers to explore the methods and to enable the reproducibility of my experimental results.

1.2 Achievements and novel contributions

The contributions of the work are as follows:

- I demonstrate the benefit of modelling heterogeneous data using Sheaf Graph Neural Networks (SheafGNNs) and Sheaf Hypergraph Neural Networks (SheafHNNs) on a series of heterogeneous datasets. These techniques are competitive against existing approaches and achieve state-of-the-art results on several benchmarks.
- I introduce a series of novel heterogeneous sheaf learners demonstrating how the restriction maps may be modified to account for the type of information encoded in the heterogeneous data.
- I propose a general message passing framework for sheafs that may be applied to graphs (Sheaf-MPNN) or hypergraphs (SheafAllSet). Additionally, I introduce SheafDeepSet for graphs that universally approximates a SheafGNN as well as SheafAllDeepSets and SheafAllSetsTransformer which universally approximates a SheafHNN. Finally, I demonstrate how this framework may be applied to any existing message passing based graph or hypergraph neural network.
- I develop the **HyperSheaf** library that provides implementations of the architectures and sheaf learners proposed in the project. This library is designed to allow others to easily reproduce the experimental results included in this report and adapt the architectures for custom datasets.

1.3 Project structure

Chapter 2: Background. This chapter covers the necessary background that is built upon in later chapters. It begins with a discussion of message passing approaches for GNNs, then continues to give a formal definition of a cellular sheaf attached to a graph. Next, it moves onto recent SheafGNN architectures proposed in literature. The chapter concludes with a discussion on how these approaches can be lifted to create Hypergraph Neural Networks (HNNs) and SheafHNNs.

Chapter 3: Related work. Discusses various homogeneous and heterogeneous graph and hypergraph architectures proposed in recent literature.

Chapter 4: Heterogeneous sheaf learners. Demonstrates how sheaf learners may be modified to account for type information in a heterogeneous setup. It also provides the time complexities of all the proposed modified sheaf learners.

Chapter 5: Experimental results. Provides a series of experimental results demonstrating the performance benefit of using sheaves for heterogeneous data.

Chapter 6: A general sheaf message passing framework. Proposes a general message passing framework for sheaves attached to graphs and hypergraphs. In addition, it proposes a series of more expressive and general architectures than those presented in the literature.

Chapter 2

Background

This chapter covers the background that underpins the work presented in subsequent chapters. It begins with a discussion of heterogeneous and homogeneous graphs and graph message passing. It then discusses how cellular sheaves may be attached to graphs and incorporated into graph-based processing techniques. The chapter then discusses how these approaches may be lifted to hypergraphs.

2.1 Homogenous and heterogenous graphs

Graphs can be classified as either homogeneous or heterogeneous. In homogeneous graphs each node or edge has the same type, whereas heterogeneous graphs have different node and edge types. Graphs can be heterogeneous in either their nodes, edges, or both. Figure 2.1 illustrates heterogeneous and homogeneous graphs. We give a formal definition below.

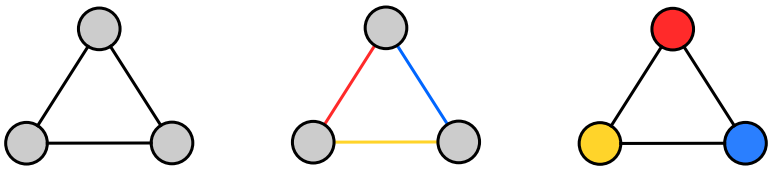


Figure 2.1: **Homogeneous and heterogeneous graphs.** **(LEFT)** A homogeneous graph. **(CENTRE)** A graph with heterogeneous edges. **(RIGHT)** A graph that is heterogeneous with respect to its nodes. Colour is used to indicate different edge types and node types, respectively.

Definition 2.1. A *heterogeneous graph* is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{T})$ where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of edges, $\mathcal{S}$ is a set of node types and $\mathcal{T}$ is a set of edge types. The type of a node $u \in \mathcal{V}$ is denoted as τ_u and the type of an edge $e \in \mathcal{E}$ as τ_e or τ_{uv}.

Remark. For the sake of convenience, we shall assume that each type is represented as an integer label, i.e. $\tau_u \in \{1, \dots, |\mathcal{S}|\}$ and $\tau_e \in \{1, \dots, |\mathcal{T}|\}$.

A real-world example of a homogeneous graph is a social network in which all the nodes represent people, and the edges represent relationships between people. The social network could be converted into an edge-heterogeneous graph by modelling the particular type of relationship, such as whether people are friends, married, or siblings. Node-heterogeneous graphs tend also to be edge-heterogeneous. This is exemplified by citation graphs, which may store information about the authors, the publication venue, and other details of published articles. In this example, edge-heterogeneity comes from the different possible relations connecting the node types, such as 'author of' and 'published at' relationships that may be modelled in the graph.

2.2 Graph Neural Networks

Graph Neural Networks (GNNs) [23, 60] are a class of neural network architectures that operate on graphs and other relational structures with nodes and edges. Figure 2.2 illustrates a simple GNN, the hidden representations are computed by aggregating a node's neighbourhood using *message passing* [5, 22]. After this, the hidden representations are fed into a classifier based on whether the downstream task is node classification, link (edge) prediction, or graph classification.

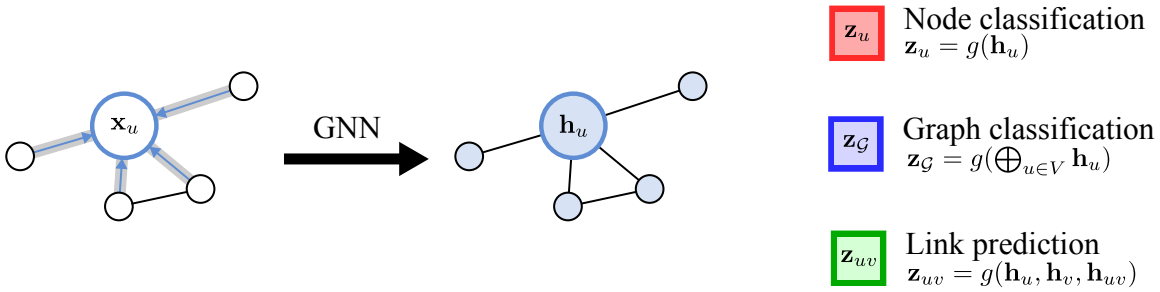


Figure 2.2: **The Graph Neural Network.** The latent representations are computed using message passing, where a node's value is updated based on the aggregation of its neighbourhood. The latent representations are then fed into a classifier based on the final downstream task. Here $\mathbf{h}_u$ is the latent representation of the node u, $\mathbf{h}_{uv}$ is the latent representation of the edge connecting node u to v and $\bigoplus$ is a permutation-invariant pooling function such as max or sum.

More fomally, we may define the GNN as a Message Passing Neural Network (MPNN) [22]. Given a graph $\mathcal{G} = (\mathcal{E}, \mathcal{V})$, a GNN layer applies the message passing over the graph to update node representations based on the aggregation of its neighbourhood. The representation of node u at layer $k+1$ is given by the following equation

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

where ϕ is the *update function* that computes the updated node representation of u, ψ is a learnable *message function* that computes the message sent from the node v to u and $\bigoplus$ is a permutation invariant aggregation function and $\mathbf{h}_u^{(0)} = \mathbf{x}_u$.

2.2.1 GNNs as modified a heat diffusion process

Recent work [6, 9] demonstrates that GNNs can be considered as a diffusion process following the heat equation, expressed using Newton's notation as

$$\dot{x} = \Delta x \tag{2.2}$$

where Δ is the Laplacian operator. Consider the graph with the adjacency matrix $\mathbf{A}$, a diagonal degree matrix $\mathbf{D}$ and a $n \times d$ node feature matrix $\mathbf{X}$. Let $\Delta_0 := \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the normalised graph Laplacian. We may now define the following heat diffusion process and its Euler discretisation with respect to time:

$$\dot{\mathbf{X}}(t) = \Delta_0 \mathbf{X}(t) \rightsquigarrow \mathbf{X}(t+1) = \mathbf{X}(t) - \Delta_0 \mathbf{X}(t) = (\mathbf{I} - \Delta_0)\mathbf{X}(t). \tag{2.3}$$

Equation (2.3) is similar to the GCN [42] update equation. A GCN is simply a modified heat diffusion equation with a learnable parameter matrix $\mathbf{W}$ and a non-linear activation σ added

$$\mathrm{GCN}(\mathbf{X}, \mathbf{A}) := \sigma\left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{W}\right) = \sigma\left((\mathbf{I} - \Delta_0)\mathbf{X}\mathbf{W}\right) \tag{2.4}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the augmented adjacency matrix with self-loops and $\tilde{\mathbf{D}}$ is the augmented degree matrix calculated using $\tilde{\mathbf{A}}$. Chamberlain et al. [9] introduced the more general framework Graph Neural Diffusion (GRAND) and demonstrated how a similar construction can be used to derive other architectures, including GAT [70] and GraphSAGE [28].

2.2.2 Oversmoothing

Chapter 1 discussed the *oversmoothing* [8] problem with MPNNs. Its effect on a heterogeneous graph is illustrated in Figure 2.3. This can be connected back to the diffusion equation above. Over time, heat equations smooth out as they reach an equilibrium. Therefore, it should not be surprising that such diffusion processes on graphs lead to increasingly smooth and similar features across node neighbourhoods. For heterogeneous graphs, this causes the features distinct to each separate node or edge type in the graph to be lost, resulting in poor performance. Due to the oversmoothing effect, modelling heterogeneous graphs using classical GNNs is challenging. This work addresses the issue by attaching a cellular sheaf to the graph to mediate the message passing process.

2.3 Cellular sheaves

Going from the photo example in Figure 1.1, let us now consider what happens if we move from images to a topological space. Let $C = \{U_\alpha : \alpha \in A\}$ be an open cover of a topological space X. The local data assignments are created by defining a space over

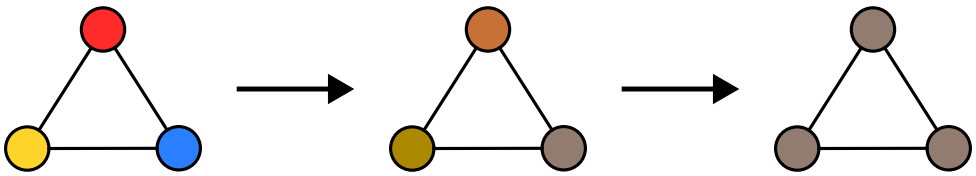


Figure 2.3: **Example of oversmoothing on a heterogeneous graph.** The arrows indicate subsequent time steps or multiple message passing layers in the network. Over time, the features become increasingly smooth and tend towards the equilibrium in the rightmost graph.

each U_α, which must be 'glued' together to create the global representation. This can be done by defining a series of maps between the spaces to connect them together. Since these maps constrain or restrict the other spaces, they are known as *restriction maps*.

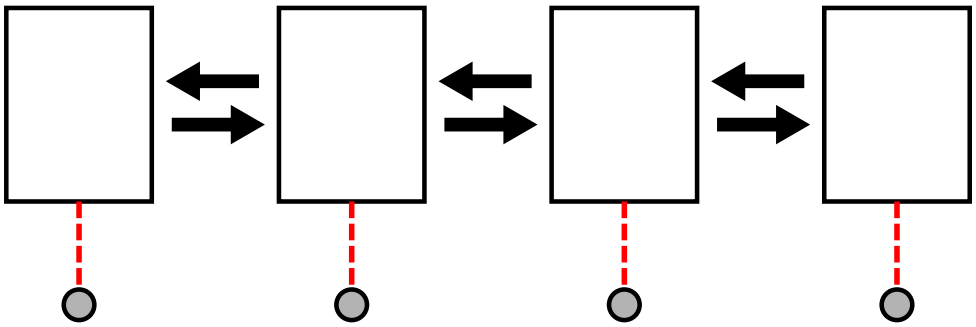


Figure 2.4: **Illustration of a sheaf attached to a set.** Here, the stalks are indicated by the white boxes attached to the element by the red dashed line. The restriction maps are indicated by black arrows.

A sheaf may also be defined over a set, which is commonly known as a *cellular sheaf* [12, 63]. Consider a group of people who are talking at a party. This can be modelled as a sheaf where the local sections are the opinions of each of the participants, and the restriction maps are the conversations between them. More formally, each element of the set has a vector space attached to it known as a *stalk* and *restriction maps* are then defined to glue the stalks together to create the global representation. This is illustrated in Figure 2.4. This can be considered a form of discrete manifold, where the points in the set are elements of the topological space, and the stalk defines the point's neighbourhood. The restriction maps then act as a form of parallel transport to connect the neighbourhoods. This construction can be applied to any set and, most importantly, for our discussion, both graphs and hypergraphs.

Definition 2.2. A *cellular sheaf* $(\mathcal{G}, \mathcal{F})$ attached to an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a triple $\langle \mathcal{F}(u), \mathcal{F}(e), \mathcal{F}_{u \trianglelefteq e} \rangle$ consisting of:

- *Node stalks* $\mathcal{F}(u)$: a vector space associated with each vertex $u \in \mathcal{V}$.
- *Edge stalk* $\mathcal{F}(e)$: a vector space for each edge $e \in \mathcal{E}$.
- Linear *restriction maps* $\mathcal{F}_{u \trianglelefteq e} : \mathcal{F}(u) \to \mathcal{F}(e)$ for each incident node-edge pair $u \trianglelefteq e$.

The space formed by the node stalks is known as the space of 0-cochains, and the space formed by the edge stalks is known as the space of 1-cochains.

Definition 2.3. Given a sheaf $(\mathcal{G}, \mathcal{F})$, the space of 0-cochains $C^0(\mathcal{G}, \mathcal{F})$ is the direct sum of the vector stalks $C^0(\mathcal{G}, \mathcal{F}) := \bigoplus_{u \in \mathcal{V}} \mathcal{F}(u)$. Similarly, the space of 1-cochains $C^1(\mathcal{G}, \mathcal{F})$ is the direct sum over the edge stalks $C^1(\mathcal{G}, \mathcal{F}) := \bigoplus_{e \in \mathcal{E}} \mathcal{F}(e)$.

Remark. For a co-chains $\mathbf{x} \in C^0(\mathcal{G}, \mathcal{F})$, $\mathbf{x}_u$ is the representation of the node u in the node stalk $\mathcal{F}(u)$.

The spaces $C^0(\mathcal{G}, \mathcal{F})$ and $C^1(\mathcal{G}, \mathcal{F})$ define the *co-boundary map* $\delta : C^0(\mathcal{G}, \mathcal{F}) \to C^1(\mathcal{G}, \mathcal{F})$. Opinion dynamics [31] provide an interpretation of each of these objects. The node stalk $\mathcal{F}(u)$ represents the 'private opinion' of the node u and $\mathcal{F}_{u \trianglelefteq e} \mathbf{x}_u$ is how the opinion is expressed in the public 'discourse space' formed by the edge stalk $\mathcal{F}(e)$. The co-boundary map δ measures the disagreement between nodes.

Definition 2.4. Given an arbitrary orientation for each edge $e = u \to v \in E$, *co-boundary map* $\delta \colon C^0(\mathcal{G}, \mathcal{F}) \to C^1(\mathcal{G}, \mathcal{F})$ is defined as $\delta(\mathbf{x})_e := \mathcal{F}_{v \trianglelefteq e} \mathbf{x}_v - \mathcal{F}_{u \trianglelefteq e} \mathbf{x}_u$.

The co-boundary map δ may be used to define the sheaf Laplacian, which measures the aggregated disagreement at each node.

Definition 2.5. The *sheaf Laplacian* of a sheaf $(\mathcal{G}, \mathcal{F})$ is defined as $L_{\mathcal{F}} := \delta^\top \delta$ and can be defined node-wise as

$$L_{\mathcal{F}}(\mathbf{x})_u = \sum_{u, v \trianglelefteq e} \mathcal{F}^\top_{u \trianglelefteq e} (\mathcal{F}_{u \trianglelefteq e} \mathbf{x}_u - \mathcal{F}_{v \trianglelefteq e} \mathbf{x}_v). \tag{2.5}$$

The sheaf Laplacian is a positive semi-definite block matrix with block diagonals of $L_{\mathcal{F}_{uu}} = \sum_{u \trianglelefteq e} \mathcal{F}^\top_{u \trianglelefteq e} \mathcal{F}_{u \trianglelefteq e}$ and off-diagonal blocks $L_{\mathcal{F}_{uv}} = -\mathcal{F}^\top_{u \trianglelefteq e} \mathcal{F}_{v \trianglelefteq e}$.

Definition 2.6. The *normalised sheaf Laplacian* is given by $\Delta_{\mathcal{F}} := D^{-1/2} L_{\mathcal{F}} D^{-1/2}$ where D is the block diagonal of $L_{\mathcal{F}}$.

Unless otherwise stated, we assign the same d-dimensional space to each stalk, that is $\mathcal{F}(u) = \mathbb{R}^d$ and $\mathcal{F}(e) = \mathbb{R}^d$ where d is the *dimension* of the stalk. This means that the sheaf Laplacian has the dimensions of $dn \times dn$. Assuming a trivial sheaf, all stalks are $\mathbb{R}$, and all restriction maps are the identity, we recover the standard graph Laplacian. This leads to a PDE analogous to Equation (2.3):

$$\mathbf{X}(0) = \mathbf{X}, \quad \dot{\mathbf{X}}(t) = -\Delta_{\mathcal{F}} \mathbf{X}(t). \tag{2.6}$$

2.3.1 Sheaves prevent oversmoothing?

Bodnar et al. [6] demonstrated that oversmoothing can be mitigated by attaching a cellular sheaf to the mediate message passing over the graph. This is due to the behaviour

of the sheaf Dirichlet energy during the diffusion process. The *Dirichlet energy* of a function f measures its variability and is minimised by the eigenvectors of the normalised Laplacian. The *Dirichlet Principle* [15, p. 42] states that solving the Laplace equation $\Delta u(x) = 0$ is equivalent to finding a function u with minimal energy subject to appropriate boundary conditions. We now define the Dirichlet energy on a graph.

Definition 2.7. The *Dirichlet energy* $E(\mathbf{x})$ of a scalar signal $\mathbf{x} \in \mathbb{R}^{|V|}$ on a (weighted) graph $\mathcal{G}$ is defined as

$$E(\mathbf{x}) = \mathbf{x}^\top \tilde{\Delta}_0 \, \mathbf{x} = \frac{1}{2} \sum_{e:=(u,v)} w_{uv} \left(\frac{x_u}{\sqrt{1+d_u}} - \frac{x_v}{\sqrt{1+d_v}} \right)^2 . \tag{2.7}$$

For a signal over a vector field $\mathbf{X} \in \mathbb{R}^{|V| \times f}$, the Dirichlet energy is defined as

$$E(\mathbf{X}) = \operatorname{tr}\left(\mathbf{X}^\top \tilde{\Delta}_0 \, \mathbf{X}\right) = \frac{1}{2} \sum_{e:=(u,v)} w_{uv} \left\| \frac{\mathbf{x}_u}{\sqrt{1+d_u}} - \frac{\mathbf{x}_v}{\sqrt{1+d_v}} \right\|_2^2 . \tag{2.8}$$

Cai and Wang [8] demonstrated that the Dirichlet energy decreases exponentially with respect to the number of layers and can be used to analyse the oversmoothing tendency of homogeneous GNNs. Minimising Equation (2.8) causes the representation of the neighbouring nodes to become similar, which is oversmoothing. So, oversmoothing can be exactly characterised in terms of the behaviour of this energy. Definition 2.6 minimises a slightly different energy function known as the *sheaf Dirichlet energy* $E_{\mathcal{F}}(\mathbf{x})$.

Definition 2.8. The *sheaf Dirichlet energy* for a sheaf $(\mathcal{G}, \mathcal{F})$ is defined as

$$E_{\mathcal{F}}(\mathbf{x}) := \mathbf{x}^\top \Delta_{\mathcal{F}} \, \mathbf{x} = \frac{1}{2} \sum_{e:=(u,v)} \left\| \mathcal{F}_{u \trianglelefteq e} \, D_u^{-1/2} \mathbf{x}_u - \mathcal{F}_{v \trianglelefteq e} \, D_v^{-1/2} \mathbf{x}_v \right\|_2^2 . \tag{2.9}$$

It is now clear why sheaves prevent the oversmoothing of node features. Equation (2.9) minimised the difference between the node representations in the edge stalk instead of the feature space, so the oversmoothing occurs in the edge stalk space instead of the node feature space. This is potentially ideal for heterogeneous setups as it will preserve the type-specific information encoded in the input features.

2.4 SheafGNNs

2.4.1 Sheaf Neural Networks

Hansen and Gebhart [30] introduced the Sheaf Neural Network based on the discrete sheaf diffusion process $\mathbf{X}(t+1) = \mathbf{X}(t) - \Delta_{\mathcal{F}} \, \mathbf{X}(t) = (\mathbf{I} - \Delta_{\mathcal{F}}) \, \mathbf{X}(t)$ based on Equation (2.6).

Assuming $\mathbf{X} \in \mathbb{R}^{nd \times f_1}$, we arrive at the following upate rule

$$\mathbf{Y} = \sigma((\mathbf{I}_{nd} - \Delta_{\mathcal{F}})(\mathbf{I}_n \otimes \mathbf{W}_1)\,\mathbf{X}\,\mathbf{W}_2) \in \mathbb{R}^{nd \times f_2}, \tag{2.10}$$

where f_1, f_2 are the number of input and output feature channels, $\otimes$ is the Kronecker product, $\mathbf{W}_1 \in \mathbb{R}^{d \times d}, \mathbf{W}_2 \in \mathbb{R}^{f_1 \times f_2}$ are weight matrices and σ is a non-linear activation function. This approach has two limitations: (i) the sheaf is hand-crafted with a dimensionality $d = 1$ using a synthetic dataset with full-knowledge of the data generation process; and (ii) the sheaf is static and cannot evolve over time based on the intermediate feature representations. *Neural Sheaf Diffusion* resolves both of these limitation.

2.4.2 Neural Sheaf Diffusion

Neural Sheaf Diffusion (NSD) [6] uses the modified diffusion equation

$$\dot{\mathbf{X}}(t) = -\sigma\Big(\Delta_{\mathcal{F}(t)}(\mathbf{I} \otimes \mathbf{W}_1)\,\mathbf{X}(t)\,\mathbf{W}_2\Big). \tag{2.11}$$

Bodnar et al. focus on a discrete version of Equation (2.11)

$$\mathbf{X}^{(t+1)} = \mathbf{X}_t - \sigma\Big(\Delta_{\mathcal{F}(t)}\Big(\mathbf{I} \otimes \mathbf{W}_1^{(t)}\Big)\,\mathbf{X}^{(t)}\,\mathbf{W}_2^{(t)}\Big). \tag{2.12}$$

where the matrix $\mathbf{X}_0 \in \mathbb{R}^{nd \times f}$ is constructed by applying an MLP to the original node feature matrix. The key difference from Hansen and Gebhart [30] is that the sheaf Laplacian $\Delta_{\mathcal{F}(t)}$ is that of a sheaf $(\mathcal{G}, \mathcal{F}(t))$ that evolves over time and can be learnt from the underlying graph structure. This allows the model to use the intermediate feature representations to update the graph's underlying topology and the diffusion process's behaviour.

Sheaf learning. Each $d \times d$ restriction map $\mathcal{F}_{u \trianglelefteq e := (u,v)}$ is learnt using a parametric function $\Phi(\mathbf{x}_u, \mathbf{x}_v)$ which is commonly a Multi-Layer Perceptron (MLP) followed by reshaping the output as it can learn any sheaf on a graph [6, Proposition 18]. It is possible to distinguish between several different functions Φ based on the types of matrices learnt:

- *Diagonal* (Diag-nsd). The restriction maps are diagonal matrices. This limits the number of learnable parameters per edge and leads to a block diagonal sheaf Laplacian, resulting in fewer operations in sparse matrix multiplication. However, a downside is that d-dimension stalks only interact with the left $\mathbf{W}_1$ multiplication.
- *Orthogonal* (O(d)-nsd). Each restriction map is an element of $O(d)$ the lie group of $d \times d$ orthogonal matrices. This effectively computes a discrete vector bundle [19, 62, 78] from differential geometry [67, Chap. 3]. Orthogonal restriction maps have several advantages: (i) they can mix stalks of various dimensions; (ii) they reduce overfitting whilst reducing the number of parameters; (iii) there is a stronger the-

oretical motivation due to its connection to parallel transport on manifolds; and (iv) the Laplacian is easier to normalise since the diagonal entries correspond to the node degrees. These orthogonal matrices are constructed from the composition of Householder reflections [50].

- *General* (Gen-nsd). Restriction maps are arbitrary matrices. This is the most flexible approach but is liable to overfitting. The sheaf Laplacian is harder to normalise numerically as $D^{-1/2}$ must be computed for each diagonal block D. The SVD must be used to perform this at scale; however, if there are repeated eigenvalues, the gradient of D will be infinite. This makes the model harder to train and more numerically unstable.

2.4.3 Sheaf Attention Networks

Barbero et al. [4] introduced Sheaf Attention Networks (SheafANs), which lift the attention mechanism of GATs to sheaves. A SheafAN layer is defined as

$$\mathbf{X}^{(t+1)} = \sigma\Big(\Big(\hat{\Lambda}\big(\mathbf{X}^{(t)}\big) \odot \hat{\mathbf{A}}_{\mathcal{F}}\Big)\Big(\mathbf{I}_n \otimes \mathbf{W}_1^{(t)}\Big)\mathbf{X}^{(t)}\,\mathbf{W}_2^{(t)}\Big) \tag{2.13}$$

where $\hat{\Lambda} = \Lambda \otimes \mathbf{1}_d$ such that $\Lambda_{u,v}(\mathbf{X}) = a(\mathbf{x}_u, \mathbf{x}_v)$ is the attention mechanism used in GATs [70], $\hat{\mathbf{A}}_{uv} = \mathbf{P}_{uv} = \mathcal{F}_{u\trianglelefteq e}^{\top}\mathcal{F}_{v\trianglelefteq e}$, $\mathbf{W}_1 \in \mathbb{R}^{d\times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{f\times f}$. Additionally, the layer may be parameterised with a residual connection to yield a Res-SheafAN layer:

$$\mathbf{X}^{(t+1)} = \sigma\Big(\Big(\hat{\Lambda}\big(\mathbf{X}^{(t)}\big) \odot \hat{\mathbf{A}}_{\mathcal{F}} - \mathbf{I}\Big)\Big(\mathbf{I}_n \otimes \mathbf{W}_1^{(t)}\Big)\mathbf{X}^{(t)}\,\mathbf{W}_2^{(t)}\Big). \tag{2.14}$$

2.5 Hypergraphs

A key limitation of graphs is that they only encode pairwise relationships. *Hypergraphs* extend graphs to consider higher-order interactions involving multiple nodes. This is done by lifting the notion of edges to hyperedges, subsets of nodes connected and interacting as a group.

Definition 2.9. A *hypergraph* is a tuple $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subset \mathcal{P}(\mathcal{V})$ is a set of hyperedges. Two nodes $u, v \in \mathcal{V}$ are connected if and only if a hyperedge $e \in \mathcal{E}$ exists such as $u, v \in e$.

Remark. Under this definition, a graph is a hypergraph with the extra constraint that all hyperedges contain exactly two nodes.

Just as graphs may be represented as an adjacency matrix, hypergraphs may be repres-

ented as a $|\mathcal{V}| \times |\mathcal{E}|$ *incidence matrix* $\mathbf{H}$ with entries defined as

$$\mathbf{H}_{ue} = \begin{cases} 1 & u \in e \\ 0 & \text{otherwise.} \end{cases} \tag{2.15}$$

Hypergraphs may also be represented as a bipartite incidence graph where each part of the graph represents the nodes and hyperedges of the hypergraph respectively with the node pair (u, e) connected if $u \in e$. Figure 2.5 illustrates both representations for an example hypergraph.

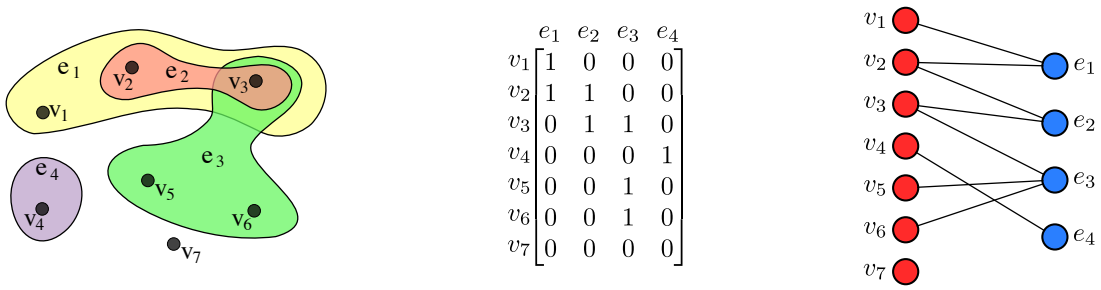


Figure 2.5: **Example hypergraph and its representations. (LEFT)** Hypergraph $\mathcal{H}$. **(CENTRE)** Incidence matrix of $\mathcal{H}$. **(RIGHT)** Incidence graph of $\mathcal{H}$. Source: Wikipedia licensed under CC BY-SA 3.0.

As with graphs, hypergraphs can be either homogeneous or heterogeneous, with the hypergraph community referring to heterogeneous hypergraphs as *multi-modal*. A hypergraph may be heterogeneous in its node, hyperedges, or both. It may be formally defined by adapting Definition 2.1 to hypergraphs.

Definition 2.10. A *heterogeneous hypergraph* is a tuple $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{T})$ where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ a set of hyperedges, $\mathcal{S}$ a set of node types, and $\mathcal{T}$ a set of hyperedge types. The type of a node $u \in \mathcal{V}$ is denoted τ_e and the type of a hyperedge $e \in \mathcal{E}$ as τ_e.

2.6 AllSet: message passing for hypergraphs

Chien et al. [10] introduced AllSet, a general message passing framework for Hypergraph Neural Networks (HNNs). The message passing occurs over the incidence graph of the hypergraph (see Figure 2.5) in two stages (Figure 2.6):

1. A message is sent from each node to its incident hyperedge and then aggregated by the hyperedge.
2. A message is sent from each hyperedge to its incident nodes and then aggregated by the node.

Let $V_{e,\mathbf{X}} = \{\{\mathbf{X}_{u,:} : u \in e\}\}$ be the multiset representation of the hidden node representations contained in the hyperedge e. Also, let $E_{e,\mathbf{Z}} = \{\{\mathbf{Z}_{e,:} : u \in e\}\}$ be the multiset

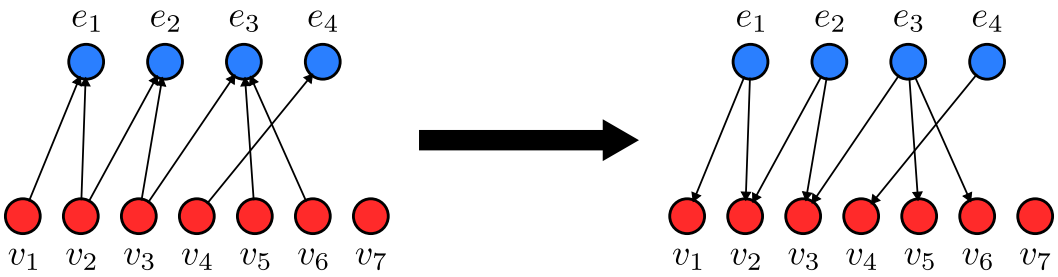


Figure 2.6: **Illustration of the AllSets framework.** Two-stage message passing applies to the hypergraph in Figure 2.5. Two multiset functions $f_{V\to E}$ and $f_{E\to V}$ are applied in sequence, both of which are permutation invariant with respect to the input multisets.

representation of the hyperedges that contain the node u where $\mathbf{Z} \in \mathbb{R}^{|\mathcal{E}|\times f'}$ is the matrix of hyperedge features. The AllSet framework has the update equation

$$\mathbf{z}_e^{(k+1)} = f_{\mathcal{V}\to\mathcal{E}}\left(V_{e,\mathbf{X}^{(t)}}; \mathbf{z}_e^{(t)}\right), \quad \mathbf{x}_u^{(k+1)} = f_{\mathcal{E}\to\mathcal{V}}\left(E_{u,\mathbf{Z}^{(t+1)}}; \mathbf{x}_u^{(t)}\right) \tag{2.16}$$

where $f_{\mathcal{V}\to\mathcal{E}}$ and $f_{\mathcal{E}\to\mathcal{V}}$ are arbitrary functions that are permutation invariant over the first argument. AllSet can be reformulated to distinguish the aggregating node from $V_{e,\mathbf{X}^{(t)}}$, giving the AllSet variant

$$\mathbf{z}_e^{(k+1),u} = f_{\mathcal{V}\to\mathcal{E}}\left(V_{e\ u,\mathbf{X}}^{(t)}; \mathbf{z}_e^{(t),u}; \mathbf{x}_u^{(t)}\right), \quad \mathbf{x}_u^{(k+1)} = f_{\mathcal{E}\to\mathcal{V}}\left(E_{u,\mathbf{Z}}^{(t+1)}; \mathbf{x}_u^{(t)}\right) \tag{2.17}$$

where the last input argument for $f_{\mathcal{V}\to\mathcal{E}}$ is ommitted unless it is explicitly required. This framework describes a series of HNNs.

HyperGNN [16] uses clique expansion to convert the hypergraph into a graph and then applies a standard GCN. The update rule can be defined node-wise as

$$\mathbf{x}_u^{(k+1)} = \sigma\left(\left[\frac{1}{\sqrt{d_u}}\sum_{e:u\in e}\frac{w_e}{|e|}\sum_{v\in e}\frac{\mathbf{x}_v^{(k)}}{d_v}\right]\Theta^{(k)} + \mathbf{b}^{(k)}\right) \tag{2.18}$$

where d_u represents the degree of node u, w_e is the predefined weight of a hyperedge and σ is non-linear activation function.

HyperGCN [76] uses a non-linear Laplacian to create partial cliques containing the most disparate nodes in the hyperedge. Its update function is defined node-wise as

$$\mathbf{x}_u^{(k+1)} = \sigma\left(\left[\sum_{e:u\in e}\sum_{v\in e} w_{uv,e}^{(k)}\,\mathbf{x}_v^{(k)}\right]\Theta^{(k)} + \mathbf{b}^{(k)}\right) \tag{2.19}$$

where the weight $w_{uv,e}^{(t)}$ depends on all of the node features in the hyperedge. Yadati et al. [76] defined the weights as

$$w_{uv,e}^{(t)} = \begin{cases} \frac{1}{2|e|-3} & \text{if } u \in \{i_e, j_e\} \text{ or } v \in \{i_e, j_e\} \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

where $(i_e, j_e) = \text{argmax}_{u,v \in e} \left\| \left(\mathbf{x}_u^{(k)} - \mathbf{x}_v^{(k)} \right) \Theta^{(k)} \right\|$.

HCHA [2] uses attentional scores based on the node-hyperedge similarity to compute a new incidence matrix. An HCHA layer may be defined node-wise as

$$\mathbf{x}_u^{(k+1)} = \sigma\left(\left[\frac{1}{\sqrt{d_u}} \sum_{e:u \in e} \frac{w_e \alpha_{ue}^{(k)}}{|e|} \sum_{v \in e} \frac{\alpha_{ve}^{(k)} \mathbf{x}_v^{(k)}}{d_v} \right] \Theta^{(k)} + \mathbf{b}^{(k)} \right). \tag{2.21}$$

This is identical to a HyperGNN layer apart from the addition of the attention weights $\alpha_{ue}^{(t)}$ and $\alpha_{ve}^{(t)}$, which are defined as

$$\alpha_{ue} = \frac{\exp(\sigma(\mathbf{a}^\top [\mathbf{x}_u \| \mathbf{z}_e]))}{\sum_{e:u \in e} \exp(\sigma(\mathbf{a}^\top [\mathbf{x}_u \| \mathbf{z}_e]))} \tag{2.22}$$

DeepAllSets [10] is more expressive and generalises the abovementioned architectures. A DeepAllSets layer is defined as the following AllSet layer

$$f_{\mathcal{V} \to \mathcal{E}}(S) = f_{\mathcal{E} \to \mathcal{V}}(S) = \text{MLP}\left(\sum_{s \in S} \text{MLP}(s) \right). \tag{2.23}$$

DeepSetTransformer [10] improves on DeepAllSets by using an attention-based mechanism to learn the relative importance of each contributing term. This is done using a modified Set Transformer [44] architecture for each multiset function. Further details on the implementation can be found in Section 4 of Chien et al. [10].

This work extends the general AllSets framework to incorporate the sheaf structure, potentially improving performance on hetereogeneous data.

2.7 Sheaf Hypergraph Neural Networks

Duta et al. [13] lifted Bodnar et al. [6] to the hypergraph domain and introduced two architectures inspired by HyperGNN [16] and HyperGCN [76], respectively. The sheaf construction for hypergraphs is similar to Definition 2.2. However, the edge stalks become *hyperedge stalks* and are instead associated with each hyperedge.

2.7.1 Sheaf hypergraph Laplacians

Duta et al. introduced two novel hypergraph Laplacian constructions: a linear Laplacian inspired by Feng et al. [16] and a non-linear Laplacian construction extending Yadati et al. [76]. The key difference is that these constructions incorporate the sheaf structure.

Definition 2.11. The *linear sheaf Laplacian* over a sheaf hypergraph $(\mathcal{F}, \mathcal{H})$ is a positive semidefinite block matrix with diagonal entries $(L_{\mathcal{F}})_{uu} := \sum_{e;u \in e} \frac{1}{\delta_e} \mathcal{F}_{u \trianglelefteq e}^\top \mathcal{F}_{u \trianglelefteq e}$ and non-

diagonal entries $(L_{\mathcal{F}})_{uv} \coloneqq -\sum_{e;u,v\in e} \frac{1}{\delta_e} \mathcal{F}^{\top}_{u\trianglelefteq e} \mathcal{F}_{v\trianglelefteq e}$.

The linear sheaf Laplacian operator applied to a node u on a signal $\mathbf{x} \in \mathbb{R}^{n\times d}$ may be written as

$$L_{\mathcal{F}}(\mathbf{x})_u \coloneqq \sum_{e;u\in e} \mathcal{F}^{\top}_{u\trianglelefteq e} \left(\sum_{\substack{v\in e \\ v\neq u}} (\mathcal{F}_{u\trianglelefteq e}\,\mathbf{x}_u - \mathcal{F}_{v\trianglelefteq e}\,\mathbf{x}_v) \right). \tag{2.24}$$

Definition 2.12. The *non-linear sheaf Laplacian* over a sheaf hypergraph $(\mathcal{F}, \mathcal{H})$ with respect to a signal $\mathbf{x} \in \mathbb{R}^{n\times d}$ is defined as:

1. For each hyperedge e compute $(u_e, v_e) \coloneqq \operatorname{argmax}_{u,v\in e} \|\mathcal{F}_{u\trianglelefteq e}\,\mathbf{x}_u - \mathcal{F}_{v\trianglelefteq e}\,\mathbf{x}_v\|$, the set of node pairs with the most discrepant features in the hyperedge stalk.
2. Build an undirected graph $\mathcal{G}_H$ with the same set of nodes as $\mathcal{H}$ where for each hyperedge e the most discrepant nodes (u, v) are connected (we shall denote this $u \sim_e v$ if they are connected due to the hyper edge e).
3. Define the non-linear sheaf Laplacian as

$$\bar{L}_{\mathcal{F}}(\mathbf{x})_u \coloneqq \sum_{e;u\sim_e v} \frac{1}{\delta_e} \mathcal{F}^{\top}_{u\trianglelefteq e} (\mathcal{F}_{u\trianglelefteq e}\,\mathbf{x}_u - \mathcal{F}_{v\trianglelefteq e}\,\mathbf{x}_v). \tag{2.25}$$

2.7.2 Sheaf Hypergraph Networks

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with a node feature matrix $\mathbf{X} \in \mathbb{R}^{n\times f}$, we initially project the input features into $\tilde{\mathbf{X}} \in \mathbb{R}^{n\times(df)}$ and then re-shape into $\tilde{\mathbf{X}} \in \mathbb{R}^{nf\times d}$. As a result, each node is represented in the vertex stalk as a $d \times f$ matrix where d is the stalk dimension and f is the number of input channels. A general Sheaf Hypergraph Network layer is defined as:

$$\mathbf{Y} \coloneqq \sigma\Big(\Big(\mathbf{I}_{nd} - \dot{\Delta}\Big)(\mathbf{I}_n \otimes \mathbf{W}_1)\tilde{\mathbf{X}}\,\mathbf{W}_2\Big), \tag{2.26}$$

where $\dot{\Delta}$ can be either $\Delta_{\mathcal{F}} = D^{-1/2} L_{\mathcal{F}} D^{-1/2}$ for the *linear* sheaf hypergraph laplacian or $\bar{\Delta}_{\mathcal{F}} = D^{-1/2} \bar{L}_{\mathcal{F}} D^{-1/2}$ in the case of the *non-linear* sheaf hypergraph Laplacian. Both $\mathbf{W}_1 \in \mathbb{R}^{d\times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{f\times f}$ are learnable parameters and σ represents ReLU non-linearity. From this, we now have two models:

- *Sheaf Hypergraph Neural Networks* (SheafHyperGNN). This model uses the linear Laplacian $\dot{\Delta} = \Delta_{\mathcal{F}}$.
- *Sheaf Hypergraph Convolutional Network* (SheafHyperGCN). This model uses the non-linear Laplacian $\dot{\Delta} = \bar{\Delta}_{\mathcal{F}}$.

Sheaf learning. As with NSD models, the restriction maps can be learnt using a parametric function. For each node-hyperedge incidence pair $u \trianglelefteq e$, the restriction maps are learnt as $\mathcal{F}_{u\trianglelefteq e} \coloneqq \text{MLP}(\mathbf{x}_u \,\|\, \mathbf{h}_e)$ where $\mathbf{x}_u$ is the stalk representation of the node u and $\mathbf{h}_e$ represents the hyperedge e. As with Bodnar et al., the learnt restriction maps can have

different properties. Duta et al. explored three types of $d \times d$ block matrices: diagonal, low-rank, and general matrices.

2.8 Topological deep learning

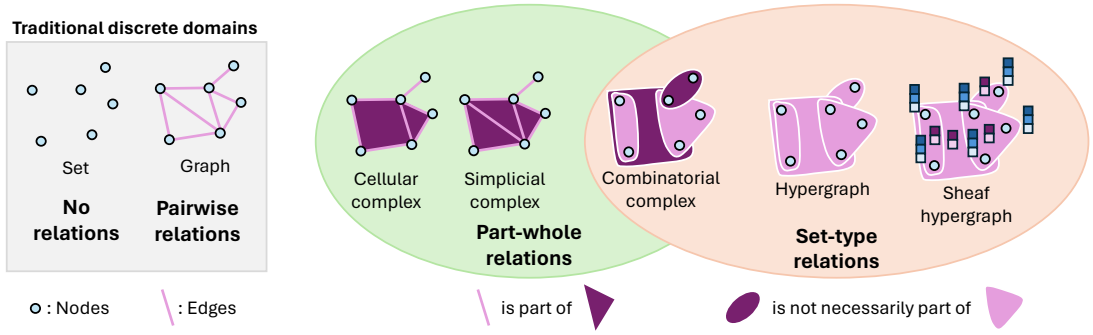


Figure 2.7: **A taxonomy of topological domains.** Adapted from Papillon et al. [53, Fig. 2]

Relational structures can be extended beyond pairwise and set-type relations modelled by graphs and hypergraphs. Papillon et al. [53] created a taxonomy of possible topological domains reproduced in Figure 2.7. Simplicial and cellular complexes extend graphs by constructing multi-scale cells to incorporate a part-whole relationship. In simplicial complexes, these cells must be simplexes with no restrictions for cellular complexes. Combinatorial complexes [26] are the most general topological structure and can be thought of as a cellular complex with the addition of hyperedges. This is used to define a general four-stage message passing framework for Topological Neural Networks (TNNs), and lift GNNs and HNNs to more general topological structures. The methods presented in this report could be incorporated into this message passing framework, allowing them to be extended to more general topological structures.

Chapter 3

Related work

This chapter summarises and discusses recent approaches for processing heterogeneous data. A discussion of homogeneous GNNs is used as a foundation to build up to heterogeneous GNN architectures. These techniques are the basis and inspiration for the sheaf modifications proposed in the next chapter.

3.1 Homogeneous GNNs

Many homogeneous GNN architectures have been proposed [29, 42, 45, 69], all of which can process a heterogeneous graph simply by ignoring the node and edge types. However, they struggle due to oversmoothing.

Graph Convolution Network (GCN) [42]. This is one of the first GNN architectures, with the l^{th} layer is defined as

$$\mathbf{H}^{(l)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}\right), \tag{3.1}$$

where $\mathbf{H}^{(l)}$ is the hidden node representations and the l^{th} layer, σ is a non-linear activation, $\mathbf{W}^{(l)}$ is a learnable weight matrix and $\tilde{\mathbf{A}}$ is the normalised adjacency matrix with added self loops.

Graph Attention Network (GAT) [69]. Instead of using a linear combination of neighbourhood features, GATs use an attentional mechanism to aggregate neighbourhood features similar to multi-head attention [68]. This means that each edge (u, v) is assigned a weight α_{uv} such that

$$\alpha_{uv} := \frac{\exp(\text{LeakyReLU}(\mathbf{a}^\top[\mathbf{W}\,\mathbf{h}_{\text{u}} \parallel \mathbf{W}\,\mathbf{h}_{\text{v}}]))}{\sum_{k\in\mathcal{N}_u}\exp(\text{LeakyReLU}(\mathbf{a}^\top[\mathbf{W}\,\mathbf{h}_{\text{u}} \parallel \mathbf{W}\,\mathbf{h}_{\text{k}}]))}. \tag{3.2}$$

where $\mathbf{W} \in \mathbb{R}^{f'\times f}$ are learnable weight parameters, $\mathbf{a} \in \mathbb{R}^{2f'}$ is a weight vector, and $\parallel$ refers to vector concatenation.

3.2 Heterogeneous GNN architectures

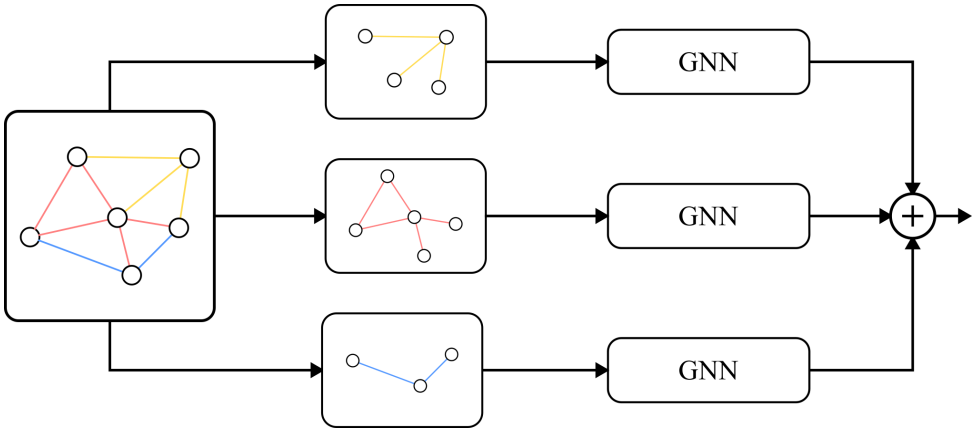


Figure 3.1: **Meta-path ensemble heterogeneous GNNs.** Illustration of a meta-path ensemble heterogeneous GNN applied to a heterogeneous graph with three edge types. The graph is split into a series of subgraphs induced by each edge type, which are fed into a separate GNN. These representations are then aggregated to generate the final latent representations.

Lv et al. [46] surveyed existing methods to process heterogeneous graphs. These methods use two techniques: (i) embed the node/relationship type as node or edge features; and (ii) an ensemble approach (Figure 3.1) based on predefined relationship sequences, known as *meta-paths*.

R-GCN [61]. Relational Graph Convolutional Networks (R-GCNs) ensemble GCNs based on edge types and aggregate the representations using a sum. The hidden representation at layer $l+1$ of a node u is a modified GCN update defined as

$$\mathbf{h}_u^{(l+1)} = \sigma\left(\sum_{t\in\mathcal{T}}\sum_{v\in\mathcal{N}_u^t}\frac{1}{c_{u,t}}\mathbf{W}_t^{(l)}\mathbf{h}_v^{(l)} + \mathbf{W}_0^{(l)}\mathbf{h}_u^{(l)}\right), \tag{3.3}$$

where $\mathcal{N}_u^t$ is the neighbourhood of a node u defined by the edge type $t \in \mathcal{T}$ and $c_{u,t}$ is a task-specific normalisation constant.

HAN [73]. Heterogeneous Attention Networks (HANs) combine node-level and semantic-level attention mechanisms. Node-level attention learns the relative importance of neighbouring nodes defined on a meta-path using a self-attention mechanism [68], which is then used to aggregate the neighbours to give a meta-path based node representation. An additional attention mechanism is used to aggregate the meta-path based node representations during semantic-level attention. So, HANs can be considered an ensemble of GATs aggregated using an attention mechanism.

HetSANN [35]. Heterogeneous Graph Structural Attention Neural Networks (HetSANNs) use a type-aware attention layer to directly encode structural and type information in a graph without explicitly exploiting meta-paths. Unfortunately, the paper provides neither details of the model hyperparameters nor any necessary preprocessing steps, so Lv et al. [46] found its performance to be poor.

HetGNN [80]. Heterogeneous Graph Neural Networks (HetGNNs) use random walks with restarts to generate neighbourhoods which are then aggregated using BiLSTM [34] modules inspired from GraphSAGE [28] to generate node features for each type and across multiple types.

HGT [36]. Heterogeneous Graph Transformers (HGTs) are inspired by transformer architectures and operate on a sampled heterogeneous subgraph. This is done in three steps: (i) *Heterogeneous Mutual Attention* based on the meta relationships; (ii) *Heterogeneous Message Passing*; and (iii) *Target-Specific Aggregation*.

Chapter 4

Heterogeneous sheaf learners

This chapter presents a series of modified sheaf learners that better account for the data's heterogeneity. The rest of the sheaf pipeline remains unchanged. The next chapter builds upon this work by demonstrating the empirical benefit of these modifications across various heterogeneous benchmarks.

4.1 Sheaves for heterogeneous data

As briefly discussed in the Introduction, even in its most basic form, a sheaf is more appropriate than standard GNNs to model heterogeneous graphs because the stalks can become specialised to specific node or edge types. The restriction maps help to learn common 'communication tunnels' between different types or modalities in the data. To some extent, the experiments in Chapter 5 show this, as over time, the restriction maps learn to implicitly encode and account for the type information.

4.2 Including type information

Standard sheaf learners (**Sheaf-NSD**) concatenate just the local features and do not explicitly account for heterogeneity in the graph or hypergraph. Instead, I wish to adapt the sheaf construction to explicitly account for the type information, which may better account for the heterogeneity and improve downstream performance. This work explores two possible approaches:

1. Embed the type information into the restriction maps as additional features to the MLP used to learn them.
2. An ensemble approach where a different parametric function is used to learn the restriction maps for each edge type.

Figure 4.1 illustrates the proposed sheaf modifications, which demonstrate different ways to include type information.

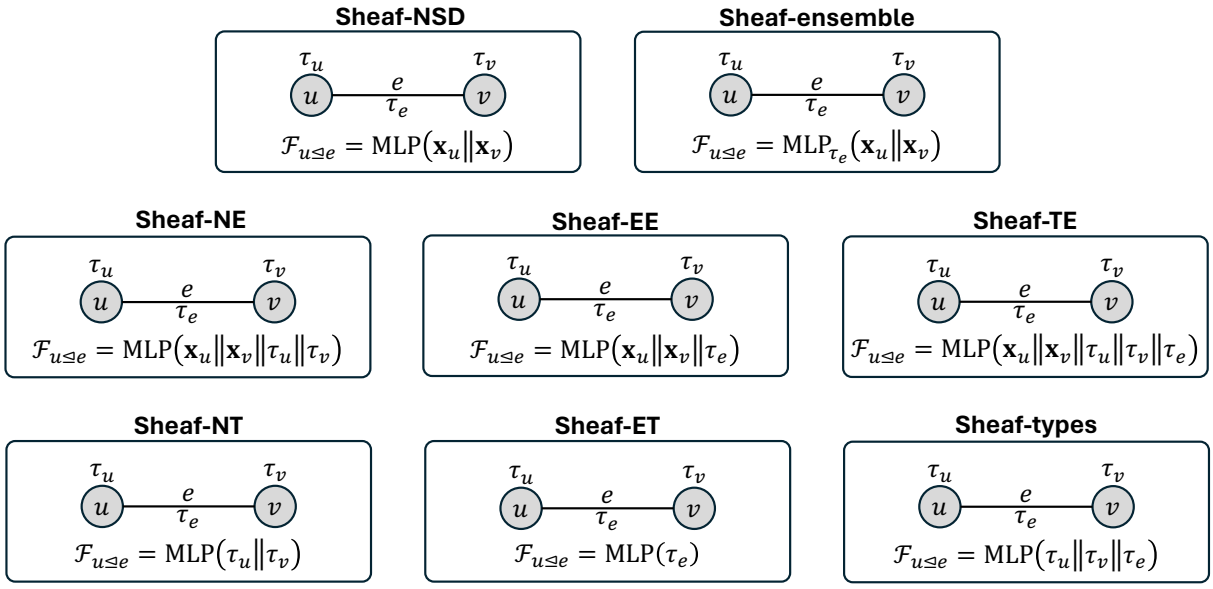


Figure 4.1: **Proposed heterogeneous sheaf learners.** Each potential sheaf modification demonstrates a different way to parameterise the restriction maps to include type information. Sheaf-NSD refers to the standard parameterisation introduced by Bodnar et al. [6] and Sheaf-ensemble an ensemble approach where a separate MLP is used for each edge type. The remaining approaches treat the type information as additional features to the MLP.

4.2.1 Embedding type information

The first approach embeds the type information as additional features in the node and edge feature vectors. As the types are categorical variables and numeric, they are encoded as a one-hot vector before they are embedded. The most straightforward approach is concatenating the one-hot encoded type vectors to the existing feature vectors. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{T})$ be a heterogeneous graph, with features $\mathbf{x}_u \in \mathbb{R}^f$ associated with each node u, $\boldsymbol{\tau}_u \in \mathbb{R}^{|\mathcal{S}|}$ be the one-hot encoded node type associated with node u, $\boldsymbol{\tau}_{uv} \in \mathbb{R}^{|\mathcal{T}|}$ be the one-hot encoded edge type associated with the edge (u, v) and $\|$ denote vector concatenation.

Sheaf-TE. Naïvely, the simplest approach would be to concatenate all the type information and the local features. This leads to the following restriction map formulation

$$\mathcal{F}_{u \trianglelefteq e} := \text{MLP}(\mathbf{x}_u \,\|\, \mathbf{x}_v \,\|\boldsymbol{\tau}_u \| \boldsymbol{\tau}_v \| \boldsymbol{\tau}_{uv}). \tag{4.1}$$

This formulation's benefit is that it explicitly accounts for all available type information, so it should, in theory, be better able to model the heterogeneity in the data than Sheaf-NSD. However, this approach is computationally wasteful in either node heterogeneity with edge homogeneity or edge heterogeneity with node homogeneity, as the model

includes redundant type information that increases the number of training parameters and could increase the chance of overfitting. This could similarly apply in cases where the edge type simply encodes the types of connected nodes, as the edge type is redundant and can be easily inferred from the node types. It may be more sensible to embed only the salient type information instead of all available type information.

Sheaf-EE. This modifies Sheaf-TE by embedding only the edge type information, giving restriction maps formulated as

$$\mathcal{F}_{u \trianglelefteq e} := \text{MLP}(\mathbf{x}_u \,\|\, \mathbf{x}_v \,\|\boldsymbol{\tau}_{uv}), \tag{4.2}$$

which provides a more natural formulation when only the graph edges are heterogeneous and is more efficient in terms of model parameters. The main downside is losing some generality and flexibility in the final computed sheaf.

Sheaf-NE. Likewise, if the graph is heterogeneous only across its nodes, it is more natural to embed just the node types. This yields restriction maps formulated as

$$\mathcal{F}_{u \trianglelefteq e} := \text{MLP}(\mathbf{x}_u \,\|\, \mathbf{x}_v \,\|\boldsymbol{\tau}_u\|\boldsymbol{\tau}_v). \tag{4.3}$$

The benefits and drawbacks of this formulation are identical to those of Sheaf-EE.

Additionally, we may learn restriction maps that include only the type information and not the local information. This is an interesting case to study for two reasons. First, it causes the sheaf to learn a message passing mechanism based only on the type information, so it is somewhat similar to how ensemble-based heterogeneous GNNs learn a message passing approach based on the edge type. The key difference is that the communication mechanism is inherent in the underlying topology of the sheaf and not the final model architecture. Secondly, it can test the relative importance of the type information compared to the local node features.

Sheaf-types. The natural starting point is to embed all available type information such that

$$\mathcal{F}_{u \trianglelefteq e} := \text{MLP}(\boldsymbol{\tau}_u\|\boldsymbol{\tau}_v\|\boldsymbol{\tau}_{uv}). \tag{4.4}$$

This approach's benefits and drawbacks are similar to those of Sheaf-TE, and it requires substantially fewer parameters than either Sheaf-TE, Sheaf-NE, or Sheaf-EE. This is, of course, assuming that $|\mathcal{S}|, |\mathcal{T}| << f$, which is a reasonable assumption in most settings. The main drawback compared to including the local features is that the learnt sheaf is likely unable to account for them and will likely perform worse than Sheaf-NSD or any of the previous approaches. As with Sheaf-TE, Sheaf-types may be modified to account for setups in which only the nodes or edges are heterogeneous.

Sheaf-ET. This approach is more natural when only the edges of the graphs are hetero-

geneous and is a modified version of Sheaf-EE. Its restriction maps are defined as

$$\mathcal{F}_{u \trianglelefteq e} := \text{MLP}(\boldsymbol{\tau}_{uv}). \tag{4.5}$$

Sheaf-NT. Finally, the restriction maps can include only the node type information, which may be useful in cases when only the nodes of the graphs are heterogeneous. The restriction maps are defined as

$$\mathcal{F}_{u \trianglelefteq e} := \text{MLP}(\boldsymbol{\tau}_u \| \boldsymbol{\tau}_v). \tag{4.6}$$

4.2.2 Ensemble-based sheaf learners

The key drawback of the approaches mentioned in the previous section is that the same MLP is used for different edge types. This implicitly assumes that the same mechanism generates the restriction maps across each edge type. The benefit of sheaves is that they can learn a different message passing mechanism for each different node or edge type through the restriction maps, so the type embedding approaches do not fully exploit this benefit. Instead, the sheaf should be able to generate the restriction maps for each edge type using a different mechanism. This may be done using a different MLP for each edge type.

Sheaf-ensemble. This uses a different MLP for each edge type. More formally, the restriction maps may be formulated as

$$\mathcal{F}_{u \trianglelefteq e} := \text{MLP}_{\tau_e}(\mathbf{x}_u \,\|\, \mathbf{x}_v) \tag{4.7}$$

where τ_e is the edge type associated with the edge $e := (u, v)$. This approach may also be augmented by concatenating the node types. The benefit of this approach is that a separate generation mechanism is used for each edge type, so it will be more expressive than reusing the same MLP across each edge type. However, it has a significant downside: it causes a substantial increase in the number of model parameters, which greatly increases the computational overhead in terms of both training time and memory usage. This could be a major issue with larger datasets, as the size of GPU memory could make this approach infeasible.

4.3 Lifting to hypergraphs

The constructions above can easily be transferred to hypergraphs by using the node and hyperedge types instead. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathcal{S}, \mathcal{T})$ be a heterogeneous hypergraph, with features $\mathbf{x}_u \in \mathbb{R}^f$ associated with each node u, $\mathbf{h}_e \in \mathbb{R}^f$ be the feature associated with each hyperedge e, $\boldsymbol{\tau}_u \in \mathbb{R}^{|\mathcal{S}|}$ be the one-hot encoded node type associated with node u,

$\boldsymbol{\tau}_e \in \mathbb{R}^{|\mathcal{T}|}$ be the one-hot encoded edge type associated with the hyperedge e. The type concatenation becomes

$$\mathcal{F}_{u \trianglelefteq e} := \text{MLP}(\mathbf{x}_u \,\|\, \mathbf{h}_e \,\|\boldsymbol{\tau}_u \| \boldsymbol{\tau}_e), \tag{4.8}$$

whilst the ensemble approach becomes

$$\mathcal{F}_{u \trianglelefteq e} := \text{MLP}_{\tau_e}(\mathbf{x}_u \,\|\, \mathbf{h}_e). \tag{4.9}$$

The other constructions proposed above may be lifted in a similar way, but they are not included for the sake of conciseness.

Computing the hyperedge features. Each proposed sheaf modification depends on the hyperedge features $\mathbf{h}_e$. Sometimes, these features are included as part of the dataset. If not, they may be computed in one of two ways:

1. Take a series of pre-computed hyperedge features as input into the model. A possible approach is to run node2vec [24] over the hypergraph incidence graph to generate the initial node and hyperedge features.
2. Inferring the hyperedge features based on aggregating the incident node features. This work uses the following approaches:
   (a) Permutation invariant aggregation of the node features such that $\mathbf{h}_e = \bigoplus_{u \in e} \mathbf{x}_u$.
   (b) Similar to Wang et al. [72], $\mathbf{h}_e = \bigoplus_{u \in e} \text{MLP}(\mathbf{x}_u)$. This has the added benefit of universally approximating all permutation equivariant hyperedge representations based on neighbourhood aggregation.
   (c) Tensor decomposition pooling [37] based on CP decomposition [33] used for tensor rank decomposition. It is defined as

$$\mathbf{h}_e = \sigma'\left(\mathbf{M}\sigma\left(\mathbf{W}^\top \begin{bmatrix}\mathbf{x}_{u_1}\\ 1\end{bmatrix} \odot \cdots \odot \begin{bmatrix}\mathbf{x}_{u_k}\\ 1\end{bmatrix}\right)\right)$$

   where $\sigma' = \text{ReLU}$ and $\sigma = \tanh$.

4.4 Computational complexity

Table 4.1 gives the complexities for each proposed sheaf modification derived following Appendix E.1 of Bodnar et al. [6]. For $1 \leqslant d \leqslant 5$ and a heterogeneous graph with a small number of node and edge types, there is an effectively constant overhead compared to GCNs and other homogeneous architectures. So, the increase in computational complexity is negligible.

Table 4.1: **Computational complexities of heterogeneous sheaf learners.** Here I assume a heterogeneous graph with n nodes, m edges, s node types and t edge types and a sheaf diffusion model with stalk dimension d and f channels such that $c = d \times f$. Diagonal restriction maps are denoted by '(diag)', and general or orthogonal restriction maps are denoted by '(non-diag)'. For the non-sheaf architectures, $c = f$ and h is the number of attention heads used by HAN or GAT.

	Computational complexity
GCN	$\mathcal{O}(nc^2 + mc)$
GAT	$\mathcal{O}(nc^2h + mch)$
HAN	$\mathcal{O}(tnc^2h + tmch)$
Sheaf-NSD (diag)	$\mathcal{O}(nc^2 + mdc)$
Sheaf-NSD (non-diag)	$\mathcal{O}(n(c^2 + d^3) + md^2(c + d))$
Sheaf-TE (diag)	$\mathcal{O}(nc^2 + md(c + s + t))$
Sheaf-TE (non-diag)	$\mathcal{O}(n(c^2 + d^3) + md^2(c + s + t + d))$
Sheaf-NE (diag)	$\mathcal{O}(nc^2 + md(c + s))$
Sheaf-NE (non-diag)	$\mathcal{O}(n(c^2 + d^3) + md^2(c + s + d))$
Sheaf-EE (diag)	$\mathcal{O}(nc^+md(c + t))$
Sheaf-EE (non-diag)	$\mathcal{O}(n(c^2 + d^3) + md^2(c + t + d))$
Sheaf-types (diag)	$\mathcal{O}(nc^2 + md(s + t))$
Sheaf-types (non-diag)	$\mathcal{O}(n(c^2 + d^3) + md^2(s + t))$
Sheaf-NT (diag)	$\mathcal{O}(nc^2 + mds)$
Sheaf-NT (non-diag)	$\mathcal{O}(n(c^2 + d^3) + md^2s)$
Sheaf-ET (diag)	$\mathcal{O}(nc^2 + mdt)$
Sheaf-ET (non-diag)	$\mathcal{O}(n(c^2 + d^3) + md^2t)$
Sheaf-ensemble (diag)	$\mathcal{O}(nc^2 + mdct)$
Sheaf-ensemble (non-diag)	$\mathcal{O}(n(c^2 + d^3) + md^2(c + dt))$

Chapter summary. This chapter introduced a series of novel sheaf learners that modify the sheaf to explicitly account for the different types. These proposed modifications incur limited computational overhead compared to standard sheaf diffusion and existing heterogeneous architectures. In the next chapter, the sheaf learners are tested on various heterogeneous tasks to explore their potential performance benefits.

Chapter 5

Experimental results

This chapter presents a series of experiments that explore the performance of sheaf-based architectures on heterogeneous graph and hypergraph tasks. The following tasks are used: (i) heterogeneous node classification; (ii) heterogeneous link prediction; and (iii) prediction of drug-target interactions using heterogeneous hypergraphs. These aim to answer the following research questions:

1. **How well do sheaf-based architectures perform compared to baseline methods?** Previous chapters have discussed the theoretical benefits of sheaf-based architectures for processing heterogeneous data compared to homogeneous GNNs. It is important to empirically verify whether this benefit exists on a series of benchmark datasets and to demonstrate the feasibility of the approach.
2. **Do the sheaf restriction maps implicitly encode edge type information?** The previous chapter theorised that sheaves can learn specialised message passing mechanisms between different types in the data. It is useful to verify this by exploring how the restriction maps of an unmodified NSD architecture change during the training process.
3. **Does embedding type information in the restriction maps improve performance?** The sheaf learners proposed in Chapter 4 explicitly account for the heterogeneity in the data and may allow the restriction maps to encode type-specific communication mechanisms. This can be empirically validated by comparing the performance of the heterogeneous sheaf learners with a standard sheaf that only concatenates the local feature information.

Each experiment was performed 10 times, with the average and standard deviation reported for each model and dataset pair. All models were implemented using PyTorch Geometric [17] and were ran on a single NVIDIA A100 with 80 GB of GPU memory. The model's hyperparameters were obtained for each architecture using a random search per-

formed using Weight & Biases sweeps[1] with the best parameters selected. Each model was trained for 200 epochs with early stopping to avoid overfitting.

5.1 Heterogeneous node classification

Datasets. I use the following benchmark datasets from the Heterogeneous Graph Benchmark (HGB) [46] node classification tasks: DBLP, IMDB and ACM. Table 5.1 provides dataset summary statistics, and Appendix A provides a more detailed description of each dataset due to space constraints.

Table 5.1: **Dataset statistics for heterogeneous node classification datasets**

	# nodes	# node types	# edges	# edge types	# classes	Task type
ACM	10 924	4	547 872	8	3	multi-class
DBLP	26 128	4	239 566	6	4	multi-class
IMDB	21 420	4	86 642	6	5	multi-label

Data splitting. The target node labels were split with 1000 labels for testing, 500 for validation, and the remaining for training.

Downstream decoder and loss function. After the GNN processes the graph, the final node embeddings are fed into an MLP to generate the final classification logits. For multiclass tasks, I use a softmax activation with cross-entropy loss, and for multilabel classification, I use a sigmoid activation with binary cross-entropy loss.

Baseline architectures. I use the standard NSD architecture with the heterogeneous sheaf learners (including the default sheaf learner) introduced in Chapter 4. They were compared against GCN, GAT, R-GCN, HAN and HGT.

Evaluation metrics. The average and standard deviation of macro- and micro-averaged F1 scores over 10 runs were reported for each model and dataset pair.

5.1.1 Results

Comparison with existing methods. Table 5.2 shows the results of the heterogeneous node classification tasks compared to various baseline architectures. The models achieve competitive results across all datasets and achieve state-of-the-art performance on both ACM and DBLP. This demonstrates the power of sheaf GNNs when processing heterogeneous data. Furthermore, all the sheaf modifications perform better than the standard local concatenation sheaf learners used by NSD, demonstrating that these modifications are useful for processing heterogeneous data.

[1] https://wandb.ai/

Table 5.2: **Performance on heterogeneous node classification.** Results of the heterogeneous sheaf learners and baselines from the literature are shown. The average macro and micro F1 score and standard deviation after 10 runs. The top three models are coloured by **First**, **Second** and **Third**.

	ACM		DBLP		IMDB	
	Macro F1	Micro F1	Macro F1	Micro F1	Macro F1	Micro F1
GAT	75.8 ± 10.7	77.91 ± 8.66	95.47 ± 0.44	95.70 ± 0.42	84.12 ± 0.96	85.31 ± 0.92
GCN	89.09 ± 3.66	89.14 ± 3.60	96.31 ± 0.73	96.57 ± 0.63	82.41 ± 1.15	83.99 ± 0.92
HAN	86.95 ± 6.19	86.64 ± 6.43	94.74 ± 0.81	95.01 ± 0.73	13.53 ± 0.24	38.70 ± 1.13
R-GCN	95.81 ± 0.39	95.75 ± 0.39	96.79 ± 0.39	97.01 ± 0.34	**88.16 ± 0.67**	**89.08 ± 0.63**
HGT	93.24 ± 3.19	93.30 ± 2.91	93.91 ± 1.08	94.26 ± 1.09	**87.74 ± 0.76**	**88.45 ± 0.71**
Sheaf-NSD	94.97 ± 0.41	94.94 ± 0.42	96.69 ± 0.82	96.89 ± 0.79	86.70 ± 0.90	87.50 ± 0.78
Sheaf-TE (ours)	96.11 ± 0.49	96.09 ± 0.51	**97.93 ± 0.36**	**98.08 ± 0.31**	86.85 ± 0.81	87.67 ± 0.80
Sheaf-ensemble (ours)	**96.16 ± 0.52**	**96.12 ± 0.54**	97.46 ± 0.64	97.62 ± 0.60	86.92 ± 1.10	87.79 ± 0.95
Sheaf-NE (ours)	**96.13 ± 0.39**	96.09 ± 0.38	97.68 ± 0.55	97.83 ± 0.51	86.87 ± 1.01	87.73 ± 0.81
Sheaf-EE (ours)	**96.39 ± 0.37**	**96.35 ± 0.36**	97.57 ± 0.69	97.73 ± 0.62	**87.12 ± 0.75**	**87.88 ± 0.67**
Sheaf-NT (ours)	96.12 ± 0.36	**96.12 ± 0.32**	**97.88 ± 0.47**	**98.04 ± 0.43**	86.92 ± 0.95	87.76 ± 0.85
Sheaf-ET (ours)	95.84 ± 0.65	95.82 ± 0.65	**97.69 ± 0.47**	**97.83 ± 0.47**	86.12 ± 0.82	87.05 ± 0.69

Impact of restriction maps. Table 5.3 shows the impact of the restriction map type on each sheaf model's performance. Across the board, general restriction maps perform the best, with orthogonal restriction maps performing the worst. This is despite the theoretical benefits of orthogonal restriction maps because of their connection with parallel transport. The most likely reason for better performance with general restriction maps is that they are more expressive and can encode more complex interactions in the edge stalks compared to the other methods. The poor performance of orthogonal restriction maps is due to the transformations they represent not being sufficiently complex enough to encode the structure and interactions in the graph datasets. Interestingly, this is in contrast to Bodnar et al. [6], which found that orthogonal restriction maps performed better on their datasets. This suggests that the best restriction map properties depend on the particular task.

For sheaves with a fixed stalk dimension d, the restriction map properties themselves encode an implicit bias about how neighbourhood features are aggregated:

- Diagonal restriction maps assume that a node's value is based on the weighted combination of its neighbours.
- Orthogonal restriction maps assume that the value of a node is defined by the sum of its neighbours. The intuition for this comes from the connection between $O(d)$ bundles and parallel transport.
- General restriction maps do not encode any assumptions about the aggregation strategy.

Impact of type information. As mentioned above, all proposed sheaf learners outperform the standard NSD model. This suggests that the type information encodes vital information required for heterogeneous tasks; it is also visible in the performance

Table 5.3: **Ablation study of the impact of different restriction map types.** The highest performing restriction map type is highlighted for each dataset and sheaf learner.

	ACM		DBLP		IMDB	
	Macro F1	Micro F1	Macro F1	Micro F1	Macro F1	Micro F1
O(d)-NSD	94.64 ± 1.02	94.59 ± 1.03	96.32 ± 0.46	96.55 ± 0.42	86.35 ± 1.29	87.20 ± 1.07
Diag-NSD	94.42 ± 0.51	94.42 ± 0.48	95.25 ± 0.70	95.52 ± 0.67	86.36 ± 0.94	87.26 ± 0.78
Gen-NSD	**94.97 ± 0.41**	**94.94 ± 0.42**	**96.69 ± 0.82**	**96.89 ± 0.79**	**86.70 ± 0.90**	**87.50 ± 0.78**
O(d)-TE	95.04 ± 0.73	95.00 ± 0.76	96.13 ± 0.74	96.37 ± 0.73	86.32 ± 0.90	87.09 ± 0.68
Diag-TE	95.74 ± 0.86	95.74 ± 0.83	97.16 ± 0.64	97.36 ± 0.62	86.32 ± 0.71	87.07 ± 0.61
Gen-TE	**96.11 ± 0.49**	**96.09 ± 0.51**	**97.93 ± 0.36**	**98.08 ± 0.31**	**86.85 ± 0.81**	**87.67 ± 0.80**
O(d)-ensemble	94.99 ± 1.23	95.00 ± 1.21	96.31 ± 0.63	96.52 ± 0.59	86.16 ± 0.71	87.03 ± 0.62
Diag-ensemble	95.89 ± 0.68	95.89 ± 0.67	96.92 ± 0.91	97.09 ± 0.89	86.62 ± 0.91	87.45 ± 0.82
Gen-ensemble	**96.16 ± 0.52**	**96.12 ± 0.54**	**97.46 ± 0.64**	**97.62 ± 0.60**	**86.92 ± 1.10**	**87.79 ± 0.95**
O(d)-NE	94.65 ± 0.99	94.64 ± 0.96	96.55 ± 0.56	96.69 ± 0.54	86.38 ± 0.93	87.11 ± 1.02
Diag-NE	94.94 ± 0.60	94.90 ± 0.59	97.01 ± 0.71	97.22 ± 0.67	86.87 ± 1.01	87.73 ± 0.81
Gen-NE	**96.13 ± 0.39**	**96.09 ± 0.38**	**97.68 ± 0.55**	**97.83 ± 0.51**	**86.54 ± 0.71**	**87.46 ± 0.74**
O(d)-EE	95.00 ± 0.80	95.00 ± 0.79	96.51 ± 0.64	96.73 ± 0.56	87.12 ± 0.75	87.88 ± 0.67
Diag-EE	95.09 ± 0.51	95.05 ± 0.51	96.86 ± 0.47	97.07 ± 0.47	85.99 ± 0.63	86.85 ± 0.58
Gen-EE	**96.39 ± 0.37**	**96.35 ± 0.36**	**97.57 ± 0.69**	**97.73 ± 0.62**	**86.60 ± 0.93**	**87.51 ± 0.92**
O(d)-NT	94.30 ± 1.10	94.28 ± 1.12	96.63 ± 0.44	96.78 ± 0.42	86.35 ± 1.04	87.26 ± 0.94
Diag-NT	94.98 ± 0.58	94.96 ± 0.60	96.96 ± 0.41	97.11 ± 0.36	85.85 ± 1.07	86.78 ± 0.91
Gen-NT	**96.12 ± 0.36**	**96.12 ± 0.32**	**97.88 ± 0.47**	**98.04 ± 0.43**	**86.92 ± 0.95**	**87.76 ± 0.85**
O(d)-ET	94.37 ± 1.35	94.32 ± 1.37	96.15 ± 0.46	96.33 ± 0.46	85.92 ± 0.68	87.01 ± 0.57
Diag-ET	95.44 ± 0.37	95.40 ± 0.38	97.34 ± 0.56	97.55 ± 0.49	85.96 ± 0.74	86.90 ± 0.69
Gen-ET	**95.84 ± 0.65**	**95.82 ± 0.65**	**97.69 ± 0.47**	**97.83 ± 0.47**	**86.12 ± 0.82**	**87.05 ± 0.69**

gap between the homogeneous and heterogeneous GNNs. For these particular datasets, Table 5.2 suggests that the type information is more informative than the node features, such as sheaves that learn the restriction maps using just the type information (Sheaf-NT and Sheaf-ET), and none of the local features perform better than those using just the local features (Sheaf-NSD).

Do restriction maps encode edge types? To explore the learnt restriction maps, I performed linear probing [66, 70] to see if they learn to encode the edge type. This was done by feeding the restriction maps of an unmodified NSD model into a simple linear classifier to classify the type of edge to which the stalk is attached. The results are shown in Figure 5.1 with the classifier accuracy plotted as a function of the number of training steps. As training progresses, the classifier accuracy increases; however, the accuracy is around 30 % to 45 %, suggesting that embeddings are poor in terms of their representation and encoding of the edge type information.

Computational overhead. Table 5.4 shows the sheaf modifications' average run time and model size compared to baseline architectures. The runtime is provided for completeness and to give an indication of the computational requirements of the models. However, it is important to note that the sheaf architectures were implemented by adapting the original NSD implementation [6]. So, the sheaf implementations could be more computationally efficient, while the baseline implementations are from PyG [17] and are substantially more optimised. The baseline architectures that performed best

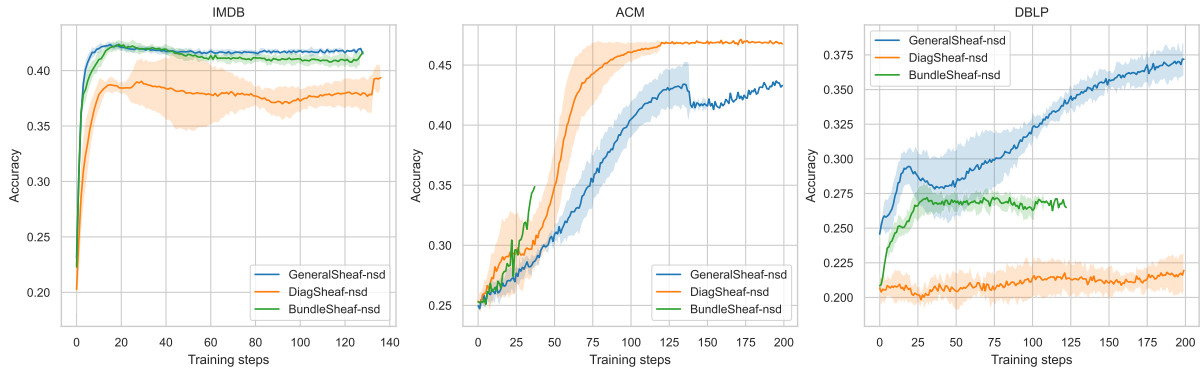


Figure 5.1: **Linear probing of the learnt restriction maps based on edge types.** The plots show the performance of a linear classifier that predicts the edge types from the learnt restriction maps of a NSD model. During the training process, the accuracy of the classifier increases, which suggests that they implicitly learn to encode the edge types.

across all datasets were HGT and R-GCN, with the largest sheaf architecture (Sheaf-ensemble) being, on average, 111x smaller than R-GCN and 17x smaller than HGT in terms of the number of model parameters across all datasets. This demonstrates the benefit of these sheaves as they either exceed or perform similarly to existing approaches with a substantially smaller model architecture and reduced memory overhead.

Table 5.4: **Computational overhead of sheaf learners.** The baseline architectures with the highest performance are highlighted.

	ACM		DBLP		IMDB	
	Runtime (s)	# params	Runtime (s)	# params	Runtime (s)	# params
GAT	111.1	8.6M	32.5	13.4M	22.7	11.9M
GCN	17.9	619K	10.4	1.2M	7.9	1.0M
HAN	28.9	2.2M	18.5	1.9M	15.8	3.4M
R-GCN	**22.1**	**43M**	**30.3**	**87.8M**	**25.9**	**72.1M**
HGT	**1025.8**	**7.2M**	**648.3**	**6.2M**	**939.9**	**10.5M**
Sheaf-NSD	47.6	155K	73.3	1.7M	40.5	1.6M
Sheaf-TE (ours)	44.7	155K	79.1	1.7M	40.1	1.6M
Sheaf-ensemble (ours)	55.7	174K	83.2	1.8M	44.9	1.9M
Sheaf-NE (ours)	47.3	155K	89.6	1.7M	12.0	1.1M
Sheaf-EE (ours)	49.6	155K	85.9	1.7M	41.8	1.6M
Sheaf-NT (ours)	42.0	154K	82.4	1.7M	38.6	1.6M
Sheaf-ET (ours)	45.1	154K	80.5	1.7M	39.8	1.6M

5.2 Heterogeneous link prediction

Datasets. The following link prediction datasets from HGB were formulated as a binary classification task: LastFM [18] and MovieLens[2]. Table 5.5 provides dataset summary statistics, and Appendix A provides a more detailed description of each dataset due to space constraints.

[2] https://movielens.org/

Table 5.5: **Dataset statistics for heterogeneous link prediction datasets**

	# nodes	# node types	# edges	# edge types	# classes
MovieLens	10 352	2	201 672	2	1
LastFM	20 612	3	378 382	9	1

Data splitting. To prevent test leakage, a random link split was performed on the dataset, which converts the original graph into a separate train, validation and test graph with 70 % of edges used for training, 10 % for validation and 20 % for testing.

Downstream decoder and loss function. For each node pair u, v and relation type r, the model outputs the score

$$P(u \sim_r v) = \sigma(\text{MLP}(\mathbf{x}_u \,\|\, \mathbf{x}_v)) \tag{5.1}$$

which gives the probability that u and v are connected by a relation of type r. I employ the Bayesian Personalised Ranking (BPR) loss [32, 56] to encourage the prediction of observed edge to be higher than an unobserved one:

$$\ell_{\text{BPR}} = -\sum_{u=1}^{|V|} \sum_{i \in \mathcal{N}_u} \sum_{j \notin \mathcal{N}_u} \ln \sigma\left(\hat{\mathbf{y}}_i - \hat{\mathbf{y}}_j\right) + \lambda \|\Theta\|_2^2 \tag{5.2}$$

where Θ is the model parameters and is a regularisation term.

Baseline architectures. The baseline architectures and setup are identical to the heterogeneous node classification results.

Evaluation metrics. Each experiment was run 10 times, and the average and standard deviation of the AUROC and AUPR were reported for each model and dataset.

5.2.1 Results

Comparison to baseline methods. Table 5.6 shows the results of the heterogeneous sheaf learners in the heterogeneous link prediction tasks. Here, the sheaf architectures outperform the baseline architectures across both datasets, demonstrating the benefit of sheaves for heterogeneous tasks. All but one of the models perform significantly worse in the LastFM dataset than MovieLens, with the biggest performance gap with GAT of around 47 %. This is likely because the LastFM graph is almost double the size and has more node and edge types than MovieLens. The performance could be improved by increasing the number of training epochs.

Just as in Table 5.2, HAN substantially underperforms on both datasets with a 14 % lower performance on LastFM and a 33 % lower performance on MovieLens, and interestingly, in contrast to the other architectures, performs better on LastFM than MovieLens. This

Table 5.6: **Performance on heterogeneous link prediction benchmarks.** Results for the heterogeneous sheaf learners and baselines are shown. The table shows the average and standard deviation of the binary AUROC and AUPR scores after 10 runs with the top three models, coloured **First**, **Second** and **Third**. The runs labelled '–' were caused by an out-of-memory error of the GPU.

	LastFM		MovieLens	
	AUPR	AUROC	AUPR	AUROC
GAT	62.88 ± 0.18	50.69 ± 0.63	97.06 ± 0.24	97.47 ± 0.21
GCN	96.84 ± 0.10	96.42 ± 0.08	99.57 ± 0.03	99.51 ± 0.03
HAN	82.48 ± 3.86	78.47 ± 3.04	63.49 ± 0.14	52.06 ± 0.27
R-GCN	96.86 ± 0.07	96.97 ± 0.05	99.06 ± 0.05	99.13 ± 0.04
HGT	–	–	–	–
Sheaf-NSD	97.16 ± 0.19	96.58 ± 0.18	**99.66 ± 0.04**	**99.57 ± 0.03**
Sheaf-TE (ours)	97.71 ± 0.52	97.23 ± 0.63	99.65 ± 0.03	99.57 ± 0.04
Sheaf-ensemble (ours)	**98.21 ± 0.15**	**97.71 ± 0.18**	**99.68 ± 0.04**	**99.59 ± 0.04**
Sheaf-NE (ours)	**97.90 ± 0.68**	**97.51 ± 0.51**	99.66 ± 0.04	99.57 ± 0.04
Sheaf-EE (ours)	97.51 ± 0.44	96.91 ± 0.52	**99.67 ± 0.05**	**99.57 ± 0.05**
Sheaf-NT (ours)	**98.24 ± 0.13**	**97.80 ± 0.18**	99.61 ± 0.03	99.52 ± 0.03
Sheaf-ET (ours)	97.84 ± 0.32	97.26 ± 0.03	99.64 ± 0.03	99.54 ± 0.03

is likely because of the ensemble approach HAN uses, which increases the number of model parameters, so more data points and training epochs are required to ensure the model sufficiently converges.

Impact of the sheaf learner. In both datasets, the heterogeneous sheaf learners perform comparably or just as well as the base NSD implementation. In LastFM, they perform better than NSD, and on MovieLens embedding, just the type information performs 0.02 % to 0.05 % lower than NSD. This again illustrates the importance of the type information, which appears to be more informative than the local features. Interestingly, Sheaf-NT, which uses just the node type information to encode the restriction maps, performs the best on LastFM. This is likely an artefact of the dataset as each edge type is *type specific*. That is, they only connect nodes of certain types. As we attempt to predict the edges of a certain type, it is natural that including only the node type information performs the best, as the model must only learn which node types define that edge type. A more interesting test could use a dataset in which an edge type may connect different node types.

Effect of restriction map types. Table 5.7 shows the effect of each restriction map type on the performance of the sheaf architecture. Unlike the node classification results above, the restriction maps appear to have a much smaller effect on the overall model performance. Aligning with both Bodnar et al. [6] and Duta et al. [13], the diagonal restriction maps perform well on the NSD architecture. The results for the heterogeneous sheaf learners are more mixed, but in general, the general restriction maps seem to perform slightly better than either diagonal or orthogonal restriction maps. However,

this difference is 0.5 % compared to the performance gap of 1 % to 2 % observed with general restriction maps for node classification.

Table 5.7: **Restriction map ablation study for heterogeneous link prediction.** The highest performing restriction map type for each sheaf learner and dataset is highlighted.

	LastFM		MovieLens	
	AUPR	AUROC	AUPR	AUROC
O(d)-NSD	96.91 ± 0.02	96.35 ± 0.21	99.62 ± 0.06	99.52 ± 0.05
Diag-NSD	**97.16 ± 0.19**	**96.58 ± 0.18**	**99.66 ± 0.04**	**99.57 ± 0.03**
Gen-NSD	97.01 ± 0.13	96.44 ± 0.18	99.66 ± 0.06	99.55 ± 0.07
O(d)-TE (ours)	97.70 ± 0.61	**97.23 ± 0.63**	99.58 ± 0.05	99.49 ± 0.06
Diag-TE (ours)	97.70 ± 0.55	97.21 ± 0.64	99.63 ± 0.08	99.53 ± 0.08
Gen-TE (ours)	**97.71 ± 0.52**	97.17 ± 0.55	**99.65 ± 0.03**	**99.57 ± 0.04**
O(d)-ensemble (ours)	97.15 ± 0.11	96.46 ± 0.13	99.61 ± 0.08	99.51 ± 0.08
Diag-ensemble (ours)	97.93 ± 0.25	97.35 ± 0.28	**99.68 ± 0.04**	**99.59 ± 0.04**
Gen-ensemble (ours)	**98.21 ± 0.15**	**97.71 ± 0.18**	99.66 ± 0.06	99.57 ± 0.05
O(d)-NE (ours)	97.90 ± 0.68	97.43 ± 0.07	99.62 ± 0.05	99.52 ± 0.04
Diag-NE (ours)	97.62 ± 0.51	97.13 ± 0.53	99.66 ± 0.05	99.56 ± 0.06
Gen-NE (ours)	**98.00 ± 0.46**	**97.51 ± 0.51**	**99.66 ± 0.04**	**99.57 ± 0.04**
O(d)-EE (ours)	96.98 ± 0.17	96.42 ± 0.17	99.59 ± 0.07	99.50 ± 0.06
Diag-EE (ours)	97.40 ± 0.49	96.84 ± 0.52	**99.67 ± 0.05**	**99.57 ± 0.05**
Gen-EE (ours)	**97.51 ± 0.44**	**96.91 ± 0.52**	99.63 ± 0.05	99.51 ± 0.06
O(d)-NT (ours)	**98.24 ± 0.13**	**97.80 ± 0.18**	**99.61 ± 0.03**	**99.52 ± 0.03**
Diag-NT (ours)	97.87 ± 0.36	97.50 ± 0.39	99.60 ± 0.05	99.51 ± 0.05
Gen-NT (ours)	98.15 ± 0.19	97.69 ± 0.25	99.61 ± 0.06	99.52 ± 0.06
O(d)-ET (ours)	96.80 ± 0.29	**96.26 ± 0.24**	99.56 ± 0.05	99.45 ± 0.04
Diag-ET (ours)	97.70 ± 0.19	97.16 ± 0.18	99.62 ± 0.06	99.54 ± 0.05
Gen-ET (ours)	**97.84 ± 0.32**	97.26 ± 0.30	**99.64 ± 0.03**	**99.54 ± 0.03**

5.3 Heterogeneous drug-target interaction prediction

Datasets. There are few examples of heterogeneous hypergraph datasets. I decided to use drug-target interaction (DTI) data used by Ruan et al. [59] to predict possible interactions between a drug and possible biomolecular targets, in this case, proteins or genes. I make use of the following medical datasets: DeepDTNet [79] and KEGG [51] Table 5.8 provides dataset summary statistics, and Appendix A provides a more detailed description of each dataset due to space constraints.

Data pre-processing. Each DTI dataset consists of three hypergraph incidence matrices: (i) a drug-target incidence matrix; (ii) a drug-disease incidence matrix; and (iii) a disease-target incidence matrix. The data is preprocessed to generate a heterogeneous hypergraph using the following steps:

Table 5.8: **Dataset statistics for heterogeneous DTI datasets**

	# nodes	# node types	# hyperedges	# edge types	# classes
KEGG	5589	3	11 177	6	1
DeepDTNet	3087	3	6147	6	1

1. Combine the incidence maps (and their transpose) to create a single heterogeneous hypergraph.
2. Generate the initial node and hyperedge feature vectors with dimensionality 128 by applying node2vec [24] over the incidence graph of the heterogeneous hypergraph.

This yields a heterogeneous hypergraph with 3 node types and 6 hyperedge types. The labels are given as a positive edge index, where each pair indicates an interaction. A 70/10/20 train-test split was performed to generate the training, validation, and test datasets. During each epoch, negative sampling was used to generate negative examples to assist the training project and to prevent a class imbalance.

Downstream decoder and loss function. The final embeddings of the nodes $\mathbf{x}_d, \mathbf{x}_t$ are extracted for a given drug-target pair (d, t) and a score is calculated in the same way as the prediction of heterogeneous links. The loss function is binary cross entropy.

Baseline architectures. The heterogeneous sheaf learners were tested using a SheafHyperGNN and then compared with existing methods: HGNN [16], HCHA [2], DeepAllSets [10] and AllSetsTransformer [10].

Evaluation metrics. The mean and standard deviation of the AUROC and AUPR over 10 runs are reported for each model and dataset.

5.3.1 Results

Comparison against baselines. Table 5.9 shows the results of the sheaf learners on the heterogeneous DTI prediction datasets compared to various baseline architectures from the literature. The results are competitive across both datasets, performing third best on DeepDTNet and second best on KEGG. As with the results of node classification and link prediction, including type information in restriction maps improves the performance of the SheafHyperGNN and demonstrates the importance of this type information. The performance gap between the sheaf architectures and AllSetsTransformer is likely due to it being more expressive than SheafHyperGNN as it is a universal approximate of the AllSet framework, which a SheafHyperGNN is a special case of. However, we may have seen a better performance in SheafHyperGNN if the model had been trained for more epochs and the parameters had been hyperparameter-tuned for longer. This was not possible with the available compute resources.

Impact of restriction map types Table 5.10 demonstrates the impact of the type of re-

Table 5.9: **Performance on heterogeneous DTI prediction benchmarks.** Results for the sheaf learners and baselines from the literature are shown. The average AUROC and AUPR and standard deviation after 10 runs. The top three models are coloured by **First**, **Second** and **Third**.

	DeepDTNet		KEGG	
	AUPR	AUROC	AUPR	AUROC
AllDeepSets	91.32 ± 2.07	92.36 ± 1.39	88.46 ± 1.05	92.40 ± 0.64
AllSetsTransformer	**93.23 ± 0.63**	**93.88 ± 0.42**	**92.60 ± 0.45**	**94.89 ± 0.19**
HCHA	82.57 ± 2.10	85.96 ± 1.26	86.92 ± 0.70	88.68 ± 0.55
HGNN	**93.18 ± 0.61**	**94.58 ± 0.48**	91.70 ± 0.58	94.25 ± 0.35
SheafHyperGNN	92.12 ± 0.28	92.36 ± 0.27	92.13 ± 0.43	94.21 ± 0.23
SheafHyperGNN-TE (ours)	**92.46 ± 0.48**	**92.51 ± 0.39**	**92.48 ± 0.77**	**94.34 ± 0.04**
SheafHyperGNN-ensemble (ours)	92.29 ± 0.46	92.34 ± 0.56	**92.25 ± 0.49**	**94.27 ± 0.38**

striction map used on the final model performance. Overall, the restriction map types significantly impact the model's performance and the impact is substantially greater than the link prediction results. Due to DeepDTNet's size, only the diagonal restriction maps could be run without causing a GPU memory error. Given a sufficiently large GPU, the DeepDTNet results would be expected to follow the trend of KEGG. On KEGG, SheafHyperGNN-TE performs the best with general restriction maps, which align with the graph-based experiments. This is likely because the general restriction maps are more expressive and can better account for the interactions between nodes in the hyperedge stalks. However, SheafHyperGNN performs best with low-rank restriction maps. A possible reason is that low-rank restriction maps are more efficient in terms of parameters compared to general restriction maps, so they may be easier to optimise in this particular set-up.

Table 5.10: **Restriction map ablation study for heterogeneous DTI prediction.** Here 'Diag' refers to diagonal restriction maps, 'Gen' are general restriction maps, 'LR' are low-rank restriction maps and 'O(d)' denotes orthogonal restriction maps. Runs denoted by '–' indicate a GPU memory error. The highest performing restriction map type for each sheaf learner and dataset is highlighted.

	DeepDTNet		KEGG	
	AUPR	AUROC	AUPR	AUROC
Diag-SheafHyperGNN	**80.91 ± 0.62**	**74.50 ± 0.74**	70.85 ± 0.81	64.03 ± 0.82
Gen-SheafHyperGNN	–	–	72.47 ± 0.63	64.99 ± 0.70
LR-SheafHyperGNN	–	–	**72.81 ± 0.53**	**65.63 ± 0.63**
O(d)-SheafHyperGNN	–	–	72.45 ± 0.44	65.22 ± 0.48
Diag-SheafHyperGNN-TE	**80.62 ± 0.65**	**74.44 ± 0.77**	70.54 ± 0.63	63.85 ± 0.74
Gen-SheafHyperGNN-TE	–	–	**72.87 ± 0.50**	**65.66 ± 0.56**
LR-SheafHyperGNN-TE	–	–	72.63 ± 0.48	65.33 ± 0.63
O(d)-SheafHyperGNN-TE	–	–	72.34 ± 0.78	65.16 ± 0.68

Impact of hyperedge features. Table 5.11 shows the effect of different methods of computing the hyperedge features. The method used has limited impact compared to

changing restriction map types with only a 1 % gap in performance between all methods compared to a 2 % gap between the different restriction map types. Across both datasets, the highest performing feature computation method is 'var3', which computed the hyperedge features like Wang et al. [72] or $\mathbf{h}_e = \sum_{u \in e} \text{MLP}(\mathbf{x}_u)$. A likely reason for this is that this is the most expressive of the proposed computation method as it universally approximates all possible equivariant hyperedge features.

Table 5.11: **Impact of hyperedge feature types on model performance.** Here 'var1' uses the input hypergraph features generated by node2vec, 'var2' takes a mean of the representations of the incident nodes, 'var3' aggregates the incidence nodes using an equivariant approach similar to Wang et al. [72] and 'CP' aggregates the incident nodes using a pooling approach similar to Hua et al. [37]. The highest performing restriction map type for each sheaf learner and dataset is highlighted.

	DeepDTNet		KEGG	
	AUPR	AUROC	AUPR	AUROC
SheafHyperGNN-CP	80.95 ± 0.79	74.63 ± 1.02	72.55 ± 0.66	65.28 ± 0.67
SheafHyperGNN-var1	80.70 ± 0.69	74.11 ± 0.99	72.46 ± 0.89	**65.43 ± 1.02**
SheafHyperGNN-var2	80.51 ± 0.63	74.15 ± 0.89	72.23 ± 0.56	64.93 ± 0.66
SheafHyperGNN-var3	**81.39 ± 0.43**	**75.20 ± 0.72**	**72.59 ± 0.45**	65.37 ± 0.69
SheafHyperGNN-TE-CP	81.33 ± 0.53	74.99 ± 0.69	72.41 ± 0.50	65.13 ± 0.75
SheafHyperGNN-TE-var1	80.71 ± 0.44	74.23 ± 0.63	72.52 ± 0.62	**65.38 ± 0.82**
SheafHyperGNN-TE-var2	81.09 ± 0.84	74.68 ± 1.11	72.18 ± 0.47	64.99 ± 0.64
SheafHyperGNN-TE-var3	**81.50 ± 0.48**	**75.28 ± 0.79**	**72.52 ± 0.44**	65.24 ± 0.47

5.4 Discussion

Sheaves perform well on heterogeneous data. From Tables 5.2, 5.6 and 5.9, it is clear that the proposed heterogeneous sheaf learners achieve either state-of-the art or competitive performance across all of the benchmark statistics with an increase in model performance of up to 2 % depending on the dataset. The key limitation of the current results is the expressivity of the sheaf models. The SheafGNN architecture is based off a GCN [42], which is less expressive than an MPNN with MLPs for the update and message functions. Meanwhile, the SheafHNN is based off a HyperGNN [16], which is less expressive than a DeepAllSets or AllSetTransformer architecture. This limitation is fixed in Chapter 6, which introduces general sheaf message passing methods for graphs and hypergraphs that provide more general and expressive sheaf architectures than existing methods.

Sheaves learn type-specific communication schemes. Chapter 4 motivates the use of sheaves for heterogeneous data and argues that the restriction maps become specialised to the edge and node types in the data and learn common 'communication tunnels' between different types or modalities. Figure 5.1 demonstrates this is true, as linear

probing shows that NSD architectures implicitly encode the type information present in the data.

However, these results have two key limitations. Firstly, the results are only for Sheaf-NSD architectures and not repeated for the other proposed heterogeneous sheaf learners. This was due to time and computing resource constraints and is left as future work. Doing this would substantially improve the robustness of this observation and help to explore possible reasons behind the effect of the type information. Secondly, linear probing does not explore the underlying structure of the restriction maps. This could be explored by using either T-SNE [47] or UMAP [48] to embed the restriction maps into a lower dimensional latent space to allow for visualisation. If the restriction maps learn to encode type-specific communication mechanisms, we would expect to see restriction maps of similar edge types clustered together. Different sheaf learners could then be compared using clustering metrics such as the Rand index [55], cluster purity and completeness. In this case, cluster purity and completeness are likely to be the most useful metrics [57].

Type information improves model performance. From the results in Tables 5.2, 5.6 and 5.9, the sheaf learners proposed in Chapter 4 outperform the standard Sheaf-NSD across all datasets bar MovieLens with a statistically significant performance gap. The results suggest that the type information allows the sheaf to better account for the underlying heterogeneity in the data, supporting the motivation in Chapter 4. From the current results, it is hard to say whether the type information helps the restriction maps specialise to different types. However, from Figure 5.1 we can infer that this is likely, as Sheaf-NSD models implicitly encode the node and edge types. Verifying this would require a series of clustering and linear probing experiments on each sheaf learner, such as those discussed above and is left as future work.

Chapter summary. This chapter has sought to empirically validate the sheaf learners proposed in Chapter 4 on a series of heterogeneous benchmark datasets. The sheaf learners achieve competitive or state-of-the-art performance across all data sets, including heterogeneous hypergraphs. In the next chapter, I develop a general framework for sheaf message passing for graphs and hypergraphs. Additionally, I introduce HyperSheaf, a library containing implementations of all of the sheaf learners' architectures proposed in this report.

Chapter 6

A general sheaf message passing framework

This chapter shows how cellular sheafs can be attached to different forms of message passing-based neural architectures through a general sheaf message passing method for both graphs and hypergraphs.

6.1 Sheaf Message Passing Neural Networks

To adapt Message Passing Neural Networks to operate on a sheaf $(\mathcal{G}, \mathcal{F})$, I introduce **Sheaf Message Passing Neural Networks (Sheaf-MPNNs)**. Whereas a standard MPNN operates on the node feature vectors, a Sheaf-MPNN operates on the 0-cochain $\mathbf{x} \in C^0(\mathcal{G}, \mathcal{F})$. Until now, it has been assumed that all sheaves have the same stalk dimension or that the same space $\mathbb{R}^d$ is used for each node and edge stalk. However, this precondition is unnecessary, so any general message passing approach should account for the case where the node and edge stalks may have any arbitrary dimension.

Figure 6.1 illustrates the Sheaf-MPNN message passing approach. The update equation for a Sheaf-MPNN at layer $k+1$ is

$$\begin{aligned} \mathbf{m}_u^{(k+1)} &= \bigoplus_{u,v \trianglelefteq e} \mathcal{F}_{u \trianglelefteq e}^{\top} \psi_e \Big(\mathcal{F}_{v \trianglelefteq e} \mathbf{x}_v^{(k)}, \mathcal{F}_{u \trianglelefteq e} \mathbf{x}_u^{(k)} \Big), \\ \mathbf{x}_u^{(k+1)} &= \varphi_u \Big(\mathbf{x}_u^{(k)}, \mathbf{m}_u^{(k+1)} \Big), \end{aligned} \tag{6.1}$$

where $\psi_e : \mathcal{F}(e) \to \mathcal{F}(e)$ is the message function, $\varphi_u : \mathcal{F}(u) \to \mathcal{F}(u)$ is the update function and $\bigoplus$ is a permutation invariant aggregation function. In practice, the final argument of ψ_e is dropped unless it is required for that particular formulation (see the proof of Theorem 6.1).

The message and update functions are defined for each node stalk and edge stalk it

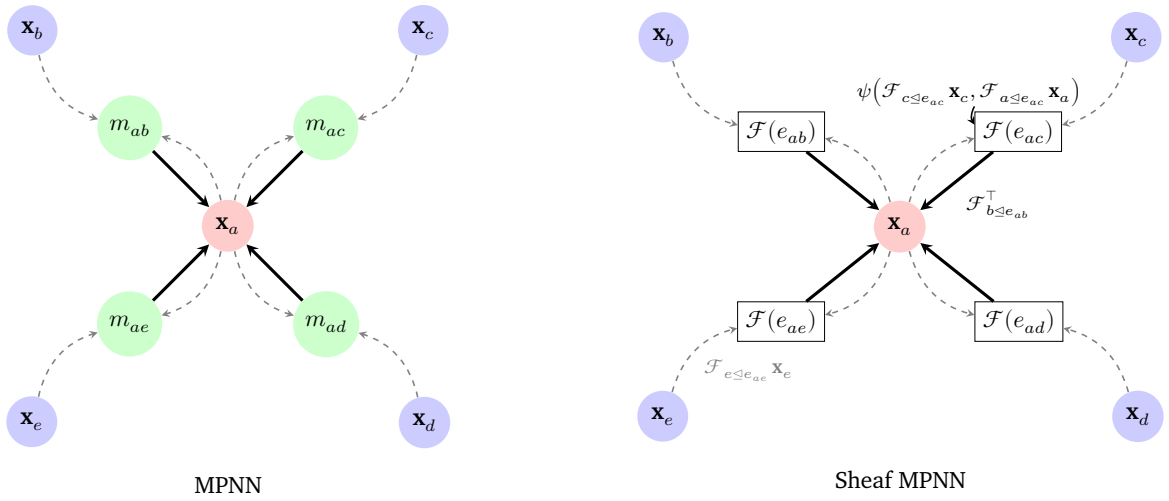


Figure 6.1: **(LEFT)**: Dataflow of a MPNN (adapted from Bronstein et al. [7]). **(RIGHT)**: Dataflow of a sheaf message passing network where the grey arrows represent the first step and the black arrows represent the final aggregation step.

operates on, as each may be an arbitrary space. In practice, the stalk dimension d is fixed to maintain tractability and to allow the message and update functions to be shared between the stalks. This yields an architecture that generalises many common sheaf neural network architectures since both Neural Sheaf Diffusion [6] and Sheaf Attention Networks [4] are special cases of Sheaf-MPNN. All proofs are provided in Appendix B.1.

Theorem 6.1. *Neural Sheaf Diffusion is a special case of Sheaf-MPNN.*

Theorem 6.2. *SheafAN and Res-SheafAN are special cases of Sheaf-MPNN.*

Theorem 6.3. *MPNNs are a special case of Sheaf-MPNN.*

6.2 SheafAllSet: sheaf message passing for hypergraphs

I propose **SheafAllSet**, which modifies AllSet [10] to operate on top of a sheaf. Let $V_e = \{\{\mathcal{F}_{u\trianglelefteq e}\mathbf{x}_u : u \in e\}\}$ be the multiset of all representations of nodes contained in the hyperedge e projected into the hyperedge stalk and $E_u = \left\{\left\{\mathcal{F}^{\top}_{u\trianglelefteq e}\mathbf{h}_e : u \in e\right\}\right\}$ be the multiset of hyperedge representations containing node u projected into the node stalk. The SheafAllSet framework uses the update rule

$$\mathbf{x}_u^{(k+1)} = f_{\mathcal{E}\to\mathcal{V}}\left(E_u^{(k+1)}; \mathbf{x}_u^{(k)}\right), \text{where } \mathbf{h}_e^{(k+1)} = f_{\mathcal{V}\to\mathcal{E}}\left(V_e^{(k)}; \mathbf{h}_e^{(k)}\right) \tag{6.2}$$

where $f_{\mathcal{V}\to\mathcal{E}}$ and $f_{\mathcal{E}\to\mathcal{V}}$ are arbitrary functions that are permutation invariant with respect to the first argument. We may also distinguish the aggregating node u from the multiset

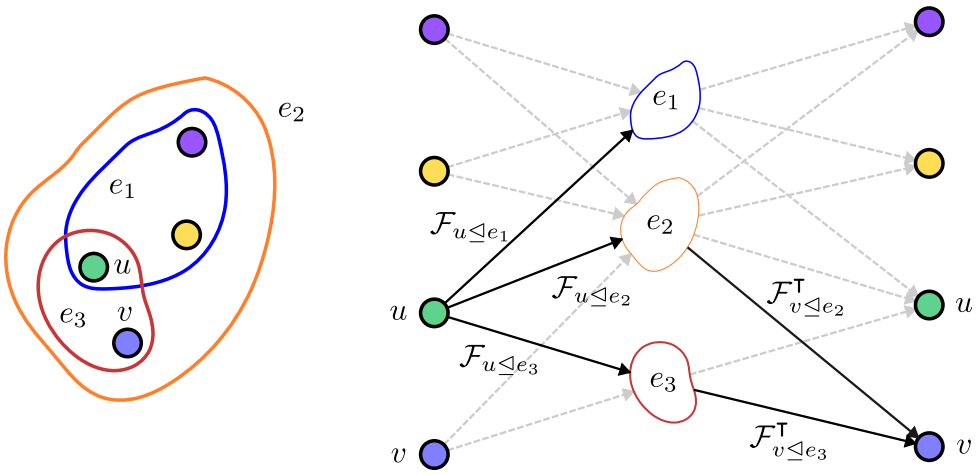


Figure 6.2: **Illustration of SheafAllSet**. Illustration of the SheafAllSet message passing approach on a hypergraph showing how the node v is updated.

V_e, giving the following SheafAllSet variant:

$$\mathbf{x}_u^{(k+1)} = f_{\mathcal{E}\to\mathcal{V}}\left(E_u^{(k+1)}; \mathbf{x}_u^{(k)}\right), \text{where } \mathbf{h}_e^{(k+1),u} = f_{\mathcal{V}\to\mathcal{E}}\left(V_{e\setminus u}^{(k)}; \mathbf{h}_e^{(k)}; \mathcal{F}_{u\trianglelefteq e}\,\mathbf{x}_u^{(k)}\right). \tag{6.3}$$

However, unless required, the last argument of $f_{\mathcal{V}\to\mathcal{E}}$ is omitted.

This is a two-stage message passing process (Figure 6.2). First, the nodes are projected onto their incidence hyperedge stalk, where they are then aggregated to compute a single hyperedge representation. The hyperedge representations are then projected back into their incidence node stalks and aggregated to produce the final message. The SheafAllSet framework gives a general approach to developing a sheaf hypergraph network and generalises both SheafHyperGNN [13] and SheafHyperGCN [13]. All proofs are provided in Appendix B.1.

Theorem 6.4. *SheafHyperGNN and SheafHyperGCN are special cases of SheafAllSet.*

Theorem 6.5. *AllSet is a special case of SheafAllSet.*

Theorem 6.6. *Sheaf-MPNN is the special case of SheafAllSet applied to graphs.*

6.3 Learning Sheaf-MPNN and SheafAllSet layers

Let $F_u = \{\{\mathbf{m}_v : v \in \mathcal{N}_u\}\}$ be a multiset represented by the batch tensor $\mathbf{F} \in \mathbb{R}^{|\mathcal{N}_u|\times d\times f}$. Following Tabaghi and Wang [64, Theorem 7], a Sheaf-MPNN can be represented as

$$\mathbf{x}_u^{(k+1)} = f(\mathbf{F}) = \rho\left(\sum_{u,v\trianglelefteq e}\phi\left(\mathcal{F}^{\top}_{u\trianglelefteq e}\,\psi_e\left(\mathcal{F}_{v\trianglelefteq e}\,\mathbf{x}_v^{(k)}\right)\right)\right), \tag{6.4}$$

where f is a permutation invariant function over $\mathbf{F}$ [see 64, Definition 4] and ρ, ϕ are some arbitrary bijective functions.

SheafDeepSet. If ϕ, ρ and ψ_e are MLPs with sufficient capacity, the resulting model is a universal approximator for Sheaf-MPNNs, termed SheafDeepSet. A SheafDeepSet layer is defined by the update equation:

$$\mathbf{x}_u^{(k+1)} = \text{MLP}\Big(\mathbf{m}_u^{(k+1)}\Big), \text{where } \mathbf{m}_u^{(k+1)} = \sum_{u,v \trianglelefteq e} \text{MLP}\Big(\mathcal{F}_{u \trianglelefteq e}^{\top} \text{MLP}\Big(\mathcal{F}_{v \trianglelefteq e} \mathbf{x}_v^{(k)}\Big)\Big). \tag{6.5}$$

As MLPs are not defined to operate on matrices, the behaviour is defined to be: (i) the input is first flattened to an $\mathbb{R}^{df}$ vector; (ii) the vector is fed through the MLP layers; and (iii) the output is reshaped back to an $\mathbb{R}^{d \times f}$ matrix.

SheafSetTransformer. Lee et al. [44] argues that the unweighted sum in DeepSet makes it difficult to learn the importance of each contributing term. To fix this, they suggest the SetTransformer architecture, which has been shown to outperform DeepSet. This may be lifted to create a SheafSetTransformer. Let the multiset

$$S_u = \Big\{\Big\{\text{Vec}\Big(\mathcal{F}_{u \trianglelefteq e}^{\top} \text{MLP}(\mathcal{F}_{v \trianglelefteq e} \mathbf{x}_v)\Big) : v \in \mathcal{N}_u\Big\}\Big\}$$

be represented as the matrix $\mathbf{S}_u \in \mathbb{R}^{|S_u| \times df}$ where $\text{Vec}(\cdot)$ refers to the vectorisation operation. This multiset may then be fed into a SetTransformer architecture [44] to give a SheafSetTransformer with the overall architecture:

$$\mathbf{Z}_u^{(k+1)} = \text{SAB}\Big(\text{SAB}\Big(\mathbf{S}_u^{(k)}\Big)\Big) \quad \mathbf{x}_u^{(k+1)} = \text{MLP}\Big(\text{SAB}\Big(\text{PMA}_1\Big(\mathbf{Z}_u^{(k+1)}\Big)\Big)\Big) \tag{6.6}$$

where SAB is set attention block [44, Sec. 3.1], PMA_k performs pooling with multihead attention [44, Sec. 3.2] to return k vectors and the MLP operates row-wise so is permutation equivariant.

Theorem 6.7. *A SheafSetTransformer is a universal approximator for Sheaf-MPNNs.*

Chien et al. [10] demonstrated that DeepAllSets and DeepSetsTransformer are universal approximators for AllSet. To construct a universal approximator for SheafAllSet, DeepAllSets and DeepSetsTransformer are lifted to sheaves, yielding SheafDeepAllSets and SheafAllSetTransformer, respectively.

SheafDeepAllSets. SheafDeepAllSets is a purely MLP SheafAllSets layer and is defined as

$$\begin{aligned} \mathbf{h}_e^{(k+1)} &= \text{MLP}\left(\sum_{u \in e} \text{MLP}\Big(\mathcal{F}_{u \trianglelefteq e} \mathbf{x}_u^{(k)}\Big)\right), \\ \mathbf{x}_u^{(k+1)} &= \text{MLP}\left(\sum_{e : u \in e} \text{MLP}\Big(\mathcal{F}_{u \trianglelefteq e}^{\top} \mathbf{h}_e^{(k+1)}\Big)\right). \end{aligned} \tag{6.7}$$

SheafAllSetTransformer. Let $\tilde{V}_e = \{\{\text{Vec}(\mathcal{F}_{u \trianglelefteq e} \mathbf{x}_u) : u \in e\}\}$ be the vetorised multiset

of V_e and $\tilde{E}_u = \left\{\!\left\{ \left\{\!\left\{ \mathrm{Vec}\left(\mathcal{F}_{u \trianglelefteq e}^{\top} \mathbf{h}_e\right) : u \in e \right\}\!\right\} \right\}\!\right\}$ the vectorised multiset of E_u. A SheafAllSetTransformer uses an AllSetTransformer [10, Eqn. 8] as each of the multiset functions and has the architecture

$$\begin{aligned} \mathbf{h}_e^{(k+1)} &= f_{\mathcal{V}\to\mathcal{E}}\left(\tilde{V}_e^{(k)}\right) = \mathrm{AllSetTransformer}\left(\tilde{V}_e^{(k)}\right) \\ \mathbf{x}_u^{(k+1)} &= f_{\mathcal{V}\to\mathcal{E}}\left(\tilde{E}_u^{(k+1)}\right) = \mathrm{AllSetTransformer}\left(\tilde{E}_u^{(k+1)}\right). \end{aligned} \tag{6.8}$$

Theorem 6.8. *SheafDeepAllSets and SheafAllSetTransformer are universal approximators of SheafAllSet.*

6.4 Attaching sheaves to MPNNs

Algorithm 1: MPNN2Sheaf-MPNN

Data: Node features $\mathbf{X}$, an MPNN with update function φ and message function ψ and a sheaf $(\mathcal{G}, \mathcal{F})$.

Result: Converts an MPNN into a Sheaf-MPNN by attaching the sheaf $(\mathcal{G}, \mathcal{F})$ with stalk dimension d.

Use an MLP to project the node features into their node stalks
foreach $u \in V$ **do**
  Compute the message $\mathbf{m}_u = \bigoplus_{u,v \trianglelefteq e} \mathcal{F}_{u \trianglelefteq e}^{\top} \psi(\mathcal{F}_{u \trianglelefteq e} \mathbf{x}_u, \mathcal{F}_{v \trianglelefteq e} \mathbf{x}_v)$
  Compute the updated node features $\mathbf{x}'_u = \varphi(\mathbf{x}_u, \mathbf{m}_u)$
end
return $\mathbf{X}'$

Algorithm 1 demonstrates how an MPNN may be converted into a Sheaf-MPNN (as long as the change in input shape is accounted for in the message and update functions) by modifying the message passing process. This can be easily implemented in any of the standard GNN libraries, such as PytorchGeometric (PyG) [17] or Deep Graph Library (DGL) [71]. In PyG, Algorithm 1 can be implemented as an adapter class to the `MessagePassing`[1] base class used to define GNN layers. In DGL, this would be done functionally using a user-defined message passing function.[2] Algorithm 2 demonstrates how this approach may be applied to an AllSet model to create a SheafAllSet model.

[1] https://pytorch-geometric.readthedocs.io/en/latest/notes/create_gnn.html

[2] See https://docs.dgl.ai/en/0.8.x/guide/message-api.html, this implementation would rely on higher-order functions that would take in a message passing function as an argument.

Algorithm 2: AllSet2SheafAllSet

Data: A hypergraph $\mathcal{H} = (E, V)$, an AllSet model with multi set functions $f_{\mathcal{E}\to\mathcal{V}}$ and $f_{\mathcal{V}\to\mathcal{E}}$, a sheaf $(\mathcal{G}, \mathcal{F})$.

Result: Converts an AllSet model into a SheafAllSetModel attaching the sheaf $(\mathcal{G}, \mathcal{F})$ with stalk dimension d.

Use an MLP to project the node features into their node stalks

`/* Message passing step 1 */`

foreach $e \in \mathcal{E}$ **do**

  Compute V_e using $\mathcal{F}$

  $\mathbf{h}_e := f_{\mathcal{V}\to\mathcal{E}}(V_e)$

end

`/* Message passing step 2 */`

foreach $u \in \mathcal{V}$ **do**

  Compute E_u using $\mathcal{F}$

  $\mathbf{x}'_u := f_{\mathcal{E}\to\mathcal{V}}(E_u)$

end

return $\mathbf{X}'$

6.5 HyperSheaf: a SheafHNN library for heterogeneous data

Previous work has demonstrated the benefit of sheaves in *heterophilic* settings where edges tend to connect dissimilar nodes. Most real-world graphs and hypergraphs are *homophilic* where edges or hyperedges tend to connect nodes with the same class or similar features, or as McPherson et al. [49] states 'birds of a feather flock together'. As a result, sheaves have been a niche technique for solving oversmoothing in heterophilic domains with few wider applications. However, this work demonstrates the benefits of sheaves for a much wider class of problems, heterogeneous graph and hypergraph data, with many wider applications far beyond those discussed in the previous chapter. To make these techniques more accessible and to aid in reproducibility, I develop **HyperSheaf**, a library for heterogeneous sheaf-based hypergraph neural networks built on top of PyTorch [54] and PyG [17].

In HyperSheaf, a heterogeneous hypergraph $\mathcal{H} = (\mathbf{X}, \mathbf{I}, \mathbf{H}, \mathbf{S}, \mathbf{T})$ is represented by a node feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|\times f}$ of $|\mathcal{V}|$ nodes, a hyperedge index $\mathbf{I}$ encoding the sparse incidence matrix in COOrdinate (COO) format, an optional hyperedge feature matrix $\mathbf{H} \in \mathbb{R}^{|\mathcal{E}|\times d}$ of $|\mathcal{E}|$ hyperedges, a vector of node types $\mathbf{S} \in \mathbb{R}^{1\times|\mathcal{V}|}$ and a vector of hyperedge types $\mathbf{T} \in \mathbb{R}^{1\times|\mathcal{E}|}$. All user-facing APIs are designed to be used in an imperative compositional style similar to PyTorch's.

Hyperedge feature builders. Implements approaches for computing hyperedge features if they do not exist during the message passing process. HyperSheaf provides the `BaseHeFeatBuilder` interface to allow users to implement custom hyperedge features.

The user only needs to define the method `compute_he_features` to use it. Listing 1 demonstrates how a custom hyperedge feature builder is implemented, in this case returning the input hyperedge features.

```
class InputFeatsHeFeatBuilder(BaseHeFeatBuilder):
    ...
    def compute_he_features(self, x, he_feats, hyperedge_index):
        row, col = hyperedge_index
        xs = torch.index_select(x, dim=0, index=row)
        es = torch.index_select(he_feats, dim=0, index=col)

        return xs, es
```

Listing 1: **Example hyperedge feature builder.** This hyperedge feature builder returns the input hyperedge features.

Heterogeneous sheaf learners. Implements different approaches to compute the sheaf restriction maps parametrically as the function

$$\mathcal{F}_{u \trianglelefteq e} = \phi(\mathbf{x}_u, \mathbf{h}_e, \tau_u, \tau_e) \tag{6.9}$$

where $\mathbf{x}_u$ is the vector representation of the node u, $\mathbf{h}_e$ the representation of the hyperedge e, τ_u deontes the type of node u and τ_e denotes the type of the hyperedge e. HyperSheaf provides the `HeteroSheafLearner` interface to implement custom sheaf learners where the user must implement the method ϕ called `predict_sheaf`. Listing 2 demonstrates how Sheaf-TE can be implemented in HyperSheaf.

```
class TypeConcatSheafLearner(HeteroSheafLearner):
    ...
    def predict_sheaf(self, node_feats, he_feats, he_index,
     ↪  node_types, he_types):
        node, hyperedge = he_index
        node_types_onehot =
         ↪  F.one_hot(node_types.to(torch.long))
        hyperedge_types_onehot =
         ↪  F.one_hot(he_types.to(torch.long))
        x_type = torch.index_select(node_types_onehot,
         ↪  dim=0, index=node)
        e_type = torch.index_select(hyperedge_types_onehot,
         ↪  dim=0, index=hyperedge)

        # sigma(MLP(x_v || h_e || t_v || t_u)))
        h_sheaf = torch.cat((node_feats, he_feats, x_type,
         ↪  e_type), dim=-1)
        h_sheaf = self.lin(h_sheaf)
        return h_sheaf
```

Listing 2: **Example heterogeneous sheaf learner.** HyperSheaf implementation of Sheaf-TE.

Models. Finally, the sheaf learners and hyperedge feature builders are used to implement SheafHNN architectures. Presently, HyperSheaf only includes implementations for a SheafHyperGNN and SheafHyperGCN with the heterogeneous sheaf learners, and hyperedge feature builders mentioned so far in the report. However, in the future, it shall be extended to include implementations of both SheafDeepAllSets and SheafAllSetTransformer.

6.6 Discussion

The theoretical results in Sections 6.1 and 6.2 demonstrate that Sheaf-MPNN and SheafAllSet generalise existing sheaf architectures. The discussion of Chapter 5 mentioned that a key limitation of the experimental results is the expressivity of the underlying sheaf architectures. This limitation has now been resolved. The modified NSD model used in the graph experiments can be replaced with either a SheafDeepSet or SheafSetTransformer model, and the SheafHyperGNN used in the hypergraph experiments can be replaced with either a SheafDeepAllSets or SheafAllSetTransformer architecture. Due to their increased expressivity, the model performance will likely increase significantly and potentially achieve state-of-the-art results.

Limitations. There are two key limitations of the framework and theoretical results presented in this chapter. Firstly, the lack of expressivity results of the Sheaf-MPNN and SheafAllSet layers presented in Section 6.3 means there are no theoretical guarantees that the layers are more expressive than existing ones. These were not added due to time constraints but can be easily derived from the existing results. Secondly, these layers in Section 6.3 have not been implemented or empirically tested. A possible future extension would be to implement these models in HyperSheaf and rerun the experiments from Chapter 5.

Chapter summary. This chapter introduced the novel Sheaf-MPNN and SheafAllSets frameworks to create general sheaf graphs and hypergraph neural networks, respectively. It then introduces a series of neural architectures that universally approximate both frameworks with SheafDeepSets and SheafSetTransformer for graphs and SheafDeepAllSets and SheafAllSetTransformer for hypergraphs. Next, it shows how any general message-passing neural network can be adapted to operate on top of a cellular sheaf, which could then be implemented into commonly used frameworks. Finally, I introduce HyperSheaf, a library for sheaf-based heterogeneous hypergraphs.

Chapter 7

Summary and conclusions

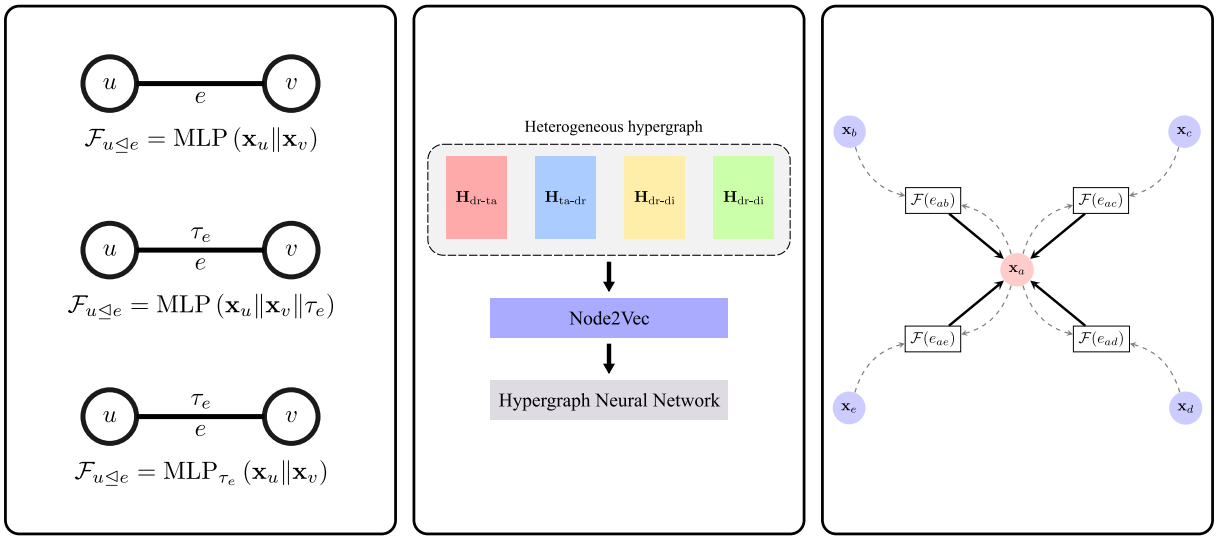
7.1 Project summary

In this project, I investigated how sheaf-based neural architectures can be used to improve performance on heterogeneous graph and hypergraph datasets. I demonstrate that by simply modifying the sheaves to account for the type information, the resulting architectures achieve competitive or state-of-the-art performance across all of the benchmark datasets. The framework presented in Chapter 6 provides a general method and approach to construct sheaf graph and hypergraph neural architectures that may be easily applied to different heterogeneous and multi-modal domains not explored in this work. I believe that the methods, general framework and HyperSheaf library presented in this report can further contribute to the wider geometric deep learning and TDL community, extending beyond the scope of this report. Figure 7.1 provides an overview of the contributions and work completed in the project.

7.2 Code availability and software engineering

In addition to the codebase used to implement the models and experiments in Chapter 5, HyperSheaf provides a useable API and interface to implement new model architectures or datasets. The library is designed with usability and familiarity in mind, and to this end, the API is inspired by PyTorch and PyG. The repository README contains install instructions and more extensive documentation. A minimal working example is provided in `tutorial.py` in the repository root. An initial version of the HyperSheaf library can be found at:

`https://github.com/AspieCoder1/HyperSheaf`



Chapter 4
Heterogeneous sheaf learners

Chapter 5
Experimental evaluation

Chapter 6
A general sheaf message passing framework

Figure 7.1: **Overview of work completed for the project. Heterogeneous sheaf learners:** I introduced a series of novel sheaf learners that take into account the type information contained in the multi-modal graph or hypergraph. **Experimental evaluation:** I performed a series of benchmarking experiments to explore the performance of the proposed heterogeneous sheaf architecture on various heterogeneous or multi-modal tasks. I find that my proposed architectures achieve state-of-the-art or competitive results on all benchmarks. **A general sheaf message passing framework:** I proposed the Sheaf-MPNN and SheafAllSet frameworks to build general SheafGNN and SheafHNN layers, as well as to introduce a series of general architectures that universally approximate both frameworks. I conclude the chapter by briefly discussing the HyperSheaf library, which I created for easily implementing multi-modal hypergraph neural networks and reproducing the results in this dissertation.

7.3 Future work

Potential future work can be broadly classified into two overarching themes:

- Adding additional functionality and documentation to make HyperSheaf more complete and user-friendly.
- Lifting the constructions presented in Chapter 6 to other topological domains using the message passing framework presented in Papillon et al. [53].

7.3.1 Extending the HyperSheaf library

Presently, the initial version of HyperSheaf only contains the implementations of a SheafHyperGNN and SheafHyperGCN with the sheaf learners discussed in this report. In the future, I intend to add both a SheafDeepAllSets and SheafAllSetsTransformer implementation to the library. Additionally, I wish to redesign the interface exposed by the library in order to make it more modular and easier to adapt. Ideally, this should be as close to 'plug and play' as possible, where it is easy to add new sheaf learners, hyperedge feature computation methods, and models in a component-based system. This would allow the library to have a declarative and module compositional style similar to that of PyTorch. Finally, the library should be properly documented and tested to ensure that it is easy to use and maintain in the long term.

7.3.2 Into domains topological

As discussed at the end of Chapter 2, Hajij et al. [26] and Papillon et al. [53] have proposed a four-stage message passing framework to generate Topological Neural Networks (TNNs). The general framework proposed in Chapter 6 can be expressed using this message passing framework, lifting sheafs into higher-order topological domains to combinatorial complexes. This would give the most general sheaf framework possible for topological and geometric domains. This sheaf framework could then be included in the Awesome TNNs [53] library and implemented as part of TopoX [27] which is a suite of tools for working with topological data and training TDL architectures in a uniform way similar to existing tooling for geometric domains.

References

[1] Unai Alvarez-Rodriguez, Federico Battiston, Guilherme Ferraz de Arruda, Yamir Moreno, Matjaž Perc and Vito Latora. 'Evolutionary dynamics of higher-order interactions in social networks'. In: *Nat Hum Behav* 5.5 (2021), pp. 586–595. ISSN: 2397-3374.

[2] Song Bai, Feihu Zhang and Philip H. S. Torr. 'Hypergraph convolution and hypergraph attention'. In: *Pattern Recognition* 110 (2021), p. 107637. ISSN: 0031-3203.

[3] Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, Michael Bronstein, Petar Veličković and Pietro Liò. 'Sheaf Neural Networks with Connection Laplacians'. In: *Proceedings of Topological, Algebraic, and Geometric Learning Workshops 2022*. Topological, Algebraic and Geometric Learning Workshops 2022. PMLR, 2022, pp. 28–36.

[4] Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde and Pietro Lio. 'Sheaf Attention Networks'. In: NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations. 2022.

[5] Peter W. Battaglia et al. *Relational inductive biases, deep learning, and graph networks*. 2018. arXiv: 1806.01261 [cs.LG]. preprint.

[6] Cristian Bodnar, Francesco Di Giovanni, Benjamin Paul Chamberlain, Pietro Lio and Michael M. Bronstein. 'Neural Sheaf Diffusion: A Topological Perspective on Heterophily and Oversmoothing in GNNs'. In: NeurIPS 2022. 2022.

[7] Michael M. Bronstein, Joan Bruna, Taco Cohen and Petar Veličković. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. 2021. arXiv: 2104.13478 [cs.LG]. preprint.

[8] Chen Cai and Yusu Wang. *A Note on Over-Smoothing for Graph Neural Networks*. 2020. arXiv: 2006.13318 [cs.LG]. preprint.

[9] Ben Chamberlain, James Rowbottom, Maria I. Gorinova, Michael Bronstein, Stefan Webb and Emanuele Rossi. 'GRAND: Graph Neural Diffusion'. In: *Proceedings of the 38th International Conference on Machine Learning*. ICML 2021. PMLR, 2021, pp. 1407–1418.

[10] Eli Chien, Chao Pan, Jianhao Peng and Olgica Milenkovic. 'You are AllSet: A Multiset Function Framework for Hypergraph Neural Networks'. In: International Conference on Learning Representations. 2021.

[11] Hejie Cui, Xinyu Fang, Ran Xu, Xuan Kan, Joyce C. Ho and Carl Yang. *Multimodal Fusion of EHR in Structures and Semantics: Integrating Clinical Records and Notes with Hypergraph and LLM*. 2024. arXiv: 2403.08818 [cs]. preprint.

[12] Justin Curry. *Sheaves, Cosheaves and Applications*. Comment: v2: 307 pages, 68 figures, includes minor edits of version 1 and is expanded with additional results, accepted as a doctoral thesis at the University of Pennsylvania. v1: 188 pages, 47 figures, work in progress. 2014. arXiv: 1303.3255 [math.AT]. preprint.

[13] Iulia Duta, Giulia Cassarà, Fabrizio Silvestri and Pietro Lio. 'Sheaf Hypergraph Networks'. In: Thirty-seventh Conference on Neural Information Processing Systems. NeurIPS 2023, 2023.

[14] Iulia Duta, Giulia Cassarà, Fabrizio Silvestri and Pietro Liò. *Sheaf Hypergraph Networks*. Comment: Accepted at Neural Information Processing Systems (NeurIPS 2023). 2023. arXiv: 2309.17116 [cs.LG]. preprint.

[15] Lawrence C Evans. *Partial differential equations*. Second edition. Graduate studies in mathematics. Providence, R.I.: American Mathematical Society, 2010. ISBN: 978-0-8218-4974-3.

[16] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji and Yue Gao. 'Hypergraph Neural Networks'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI'2019. Vol. 33. 01. 2019, pp. 3558–3565.

[17] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. Comment: ICLR 2019 (RLGM Workshop). 2019. arXiv: 1903.02428 [cs.LG]. preprint.

[18] Xinyu Fu, Jiani Zhang, Ziqiao Meng and Irwin King. 'MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding'. In: *Proceedings of The Web Conference 2020*. WWW '20. Comment: To appear at WWW 2020; 11 pages, 4 figures; typos of model name corrected. 2020, pp. 2331–2341.

[19] Tingran Gao, Jacek Brodzki and Sayan Mukherjee. 'The Geometry of Synchronization Problems and Learning Group Actions'. In: *Discrete Comput Geom* 65.1 (2021), pp. 150–211. ISSN: 1432-0444.

[20] Yue Gao, Ehsan Adeli-M, Minjeong Kim, Panteleimon Giannakopoulos, Sven Haller and Dinggang Shen. 'Medical Image Retrieval Using Multi-graph Learning for MCI Diagnostic Assistance'. In: *Med Image Comput Comput Assist Interv* 9350 (2015), pp. 86–93. pmid: 27054200.

[21] Yue Gao, Chong-Yaw Wee, Minjeong Kim, Panteleimon Giannakopoulos, Marie-Louise Montandon, Sven Haller and Dinggang Shen. 'MCI Identification by Joint Learning on Multiple MRI Data'. In: *Med Image Comput Comput Assist Interv* 9350 (2015), pp. 78–85. pmid: 26942232.

[22] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals and George E. Dahl. *Neural Message Passing for Quantum Chemistry*. Comment: 14 pages. 2017. arXiv: 1704.01212 [cs.LG]. preprint.

[23] Marco Gori, Gabriele Monfardini and Franco Scarselli. 'A new model for learning in graph domains'. In: *Proceedings. IEEE International Joint Conference on Neural Networks*. IEEE International Joint Conference on Neural Networks. Vol. 2. 2005, pp. 729–734.

[24] Aditya Grover and Jure Leskovec. 'node2vec: Scalable Feature Learning for Networks'. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 855–864. ISBN: 978-1-4503-4232-2.

[25] Tingting Guo, Yining Zhang, Yanfang Xue, Lishan Qiao and Dinggang Shen. 'Brain Function Network: Higher Order vs. More Discrimination'. In: *Front Neurosci* 15 (2021), p. 696639. ISSN: 1662-4548. pmid: 34497485.

[26] Mustafa Hajij et al. *Topological Deep Learning: Going Beyond Graph Data*. 2023. arXiv: 2206.00606 [cs.LG]. preprint.

[27] Mustafa Hajij et al. *TopoX: A Suite of Python Packages for Machine Learning on Topological Domains*. 2024. arXiv: 2402.02441 [cs.LG]. preprint.

[28] Will Hamilton, Zhitao Ying and Jure Leskovec. 'Inductive Representation Learning on Large Graphs'. In: *Advances in Neural Information Processing Systems*. NIPS 2017. Vol. 30. Curran Associates, Inc., 2017.

[29] William L. Hamilton. *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Cham: Springer International Publishing, 2020. ISBN: 978-3-031-00460-5.

[30] Jakob Hansen and Thomas Gebhart. *Sheaf Neural Networks*. Comment: NeuRips 2020 Workshop on TDA and Beyond. 2020. arXiv: 2012.06333 [cs.LG]. preprint.

[31] Jakob Hansen and Robert Ghrist. *Opinion Dynamics on Discourse Sheaves*. Comment: 23 pages, 7 figures. 2020. arXiv: 2005.12798 [math.DS]. preprint.

[32] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, YongDong Zhang and Meng Wang. 'LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation'. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 639–648. ISBN: 978-1-4503-8016-4.

[33] Frank L. Hitchcock. 'The Expression of a Tensor or a Polyadic as a Sum of Products'. In: *Journal of Mathematics and Physics* 6.1-4 (1927), pp. 164–189. ISSN: 1467-9590.

[34] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-Term Memory'. In: *Neural Comput.* 9.8 (1997), pp. 1735–1780. ISSN: 0899-7667.

[35] Huiting Hong, Hantao Guo, Yucheng Lin, Xiaoqing Yang, Zang Li and Jieping Ye. *An Attention-based Graph Neural Network for Heterogeneous Structural Learning*. 2019. arXiv: 1912.10832 [cs.LG]. preprint.

[36] Ziniu Hu, Yuxiao Dong, Kuansan Wang and Yizhou Sun. *Heterogeneous Graph Transformer*. Comment: Published on WWW 2020. 2020. arXiv: 2003.01332 [cs.LG]. preprint.

[37] Chenqing Hua, Guillaume Rabusseau and Jian Tang. 'High-order pooling for graph neural networks with tensor decomposition'. In: NeurIPS 2022. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2022, pp. 6021–6033. ISBN: 978-1-71387-108-8.

[38] Kexin Huang, Cao Xiao, Lucas M. Glass, Marinka Zitnik and Jimeng Sun. 'SkipGNN: predicting molecular interactions with skip-graph networks'. In: *Sci Rep* 10.1 (1 2020), p. 21092. ISSN: 2045-2322.

[39] Jürgen Jost and Raffaella Mulas. 'Hypergraph Laplace operators for chemical reaction networks'. In: *Advances in Mathematics* 351 (2019), pp. 870–896. ISSN: 0001-8708.

[40] Minoru Kanehisa and Susumu Goto. 'KEGG: Kyoto Encyclopedia of Genes and Genomes'. In: *Nucleic Acids Res* 28.1 (2000), pp. 27–30. ISSN: 0305-1048. pmid: 10592173.

[41] Eun-Sol Kim, Woo Young Kang, Kyoung-Woon On, Yu-Jung Heo and Byoung-Tak Zhang. 'Hypergraph Attention Networks for Multimodal Learning'. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, pp. 14581–14590.

[42] Thomas N. Kipf and Max Welling. 'Semi-Supervised Classification with Graph Convolutional Networks'. In: ICLR 2016. 2016.

[43] Craig Knox et al. 'DrugBank 6.0: the DrugBank Knowledgebase for 2024'. In: *Nucleic Acids Research* 52.D1 (2024), pp. D1265–D1275. ISSN: 0305-1048, 1362-4962.

[44] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi and Yee Whye Teh. 'Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks'. In: *Proceedings of the 36th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, 2019, pp. 3744–3753.

[45] Yujia Li, Daniel Tarlow, Marc Brockschmidt and Richard Zemel. *Gated Graph Sequence Neural Networks*. Comment: Published as a conference paper in ICLR 2016. Fixed a typo. 2017. arXiv: 1511.05493 [cs.LG]. preprint.

[46] Qingsong Lv et al. *Are we really making much progress? Revisiting, benchmarking, and refining heterogeneous graph neural networks*. Comment: KDD 2021 research track. 2021. arXiv: 2112.14936 [cs.LG]. preprint.

[47] Laurens van der Maaten and Geoffrey Hinton. 'Visualizing Data using t-SNE'. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. ISSN: 1533-7928.

[48] Leland McInnes, John Healy and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. Comment: Reference implementation available at http://github.com/lmcinnes/umap. 2020. arXiv: 1802.03426 [stat.ML]. preprint.

[49] Miller McPherson, Lynn Smith-Lovin and James M. Cook. 'Birds of a Feather: Homophily in Social Networks'. In: *Annual Review of Sociology* 27 (Volume 27, 2001 2001), pp. 415–444. ISSN: 0360-0572, 1545-2115.

[50] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman and James Bailey. 'Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections'. In: *Proceedings of the 34th International Conference on Machine Learning*. ICML 2017. PMLR, 2017, pp. 2401–2409.

[51] Sameh K Mohamed, Vít Nováček and Aayah Nounu. 'Discovering protein drug targets using knowledge graph embeddings'. In: *Bioinformatics* 36.2 (2020), pp. 603–610. ISSN: 1367-4803.

[52] Anton Obukhov. *Efficient Householder Transformation in PyTorch*. Version v1.0.1. Version: 1.0.1, DOI: 10.5281/zenodo.5068733. 2021.

[53] Mathilde Papillon, Sophia Sanborn, Mustafa Hajij and Nina Miolane. *Architectures of Topological Deep Learning: A Survey on Topological Neural Networks*. 2023. arXiv: 2304.10031 [cs.LG]. preprint.

[54] Adam Paszke et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[55] William M. Rand. 'Objective Criteria for the Evaluation of Clustering Methods'. In: *Journal of the American Statistical Association* 66.336 (1971), pp. 846–850. ISSN: 0162-1459.

[56] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner and Lars Schmidt-Thieme. 'BPR: Bayesian Personalized Ranking from Implicit Feedback'. In: (2009).

[57] Andrew Rosenberg and Julia Hirschberg. 'V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure'. In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. EMNLP-CoNLL 2007. Ed. by Jason Eisner. Prague, Czech Republic: Association for Computational Linguistics, 2007, pp. 410–420.

[58] Daniel Rosiak. *Sheaf theory through examples*. 1st ed. Cambridge, Massachusetts: MIT Press, 2022. ISBN: 978-0-262-36237-5.

[59] Ding Ruan, Shuyi Ji, Chenggang Yan, Junjie Zhu, Xibin Zhao, Yuedong Yang, Yue Gao, Changqing Zou and Qionghai Dai. 'Exploring complex and heterogeneous correlations on hypergraph for the prediction of drug-target interactions'. In: *Patterns* 2.12 (2021), p. 100390. ISSN: 2666-3899.

[60] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner and Gabriele Monfardini. 'The Graph Neural Network Model'. In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. ISSN: 1941-0093.

[61] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov and Max Welling. *Modeling Relational Data with Graph Convolutional Networks*. 2017. arXiv: 1703.06103 [stat.ML]. preprint.

[62] Luis Scoccola and Jose A. Perea. 'Approximate and discrete Euclidean vector bundles'. In: *Forum of Mathematics, Sigma* 11 (2023), e20. ISSN: 2050-5094.

[63] Allen Dudley Shepard. 'A Cellular Description of the Derived Category of a Stratified Space'. Brown University, 1985. 312 pp.

[64] Puoya Tabaghi and Yusu Wang. *Universal Representation of Permutation-Invariant Functions on Vectors and Tensors*. 2023. arXiv: 2310.13829 [cs]. preprint.

[65] Catherine Tong, Emma Rocheteau, Petar Veličković, Nicholas Lane and Pietro Liò. 'Predicting Patient Outcomes with Graph Representation Learning'. In: *AI for Disease Surveillance and Pandemic Intelligence: Intelligent Disease Detection in Action*. Ed. by Arash Shaban-Nejad, Martin Michalowski and Simone Bianco. Studies in Computational Intelligence. Cham: Springer International Publishing, 2022, pp. 281–293. ISBN: 978-3-030-93080-6.

[66] Anton Tsitsulin, Marina Munkhoeva and Bryan Perozzi. *Unsupervised Embedding Quality Evaluation*. Comment: As appeared at the 2nd Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML) at the 40th International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA. 2023. 2023. arXiv: 2305.16562 [cs.LG]. preprint.

[67] Loring W. Tu. *An introduction to manifolds*. Universitext. New York: Springer, 2008. xv+360. ISBN: 978-0-387-48098-5.

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Łukasz Kaiser and Illia Polosukhin. 'Attention is All you Need'. In: *Advances in Neural Information Processing Systems*. NIPS 2017. Vol. 30. Curran Associates, Inc., 2017.

[69] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò and Yoshua Bengio. 'Graph Attention Networks'. In: ICLR 2018. 2018.

[70] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio and R. Devon Hjelm. 'Deep Graph Infomax'. In: International Conference on Learning Representations. 2018.

[71] Minjie Wang et al. *Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks*. Comment: Major update with significantly more results. 2020. arXiv: 1909.01315 [cs.LG]. preprint.

[72] Peihao Wang, Shenghao Yang, Yunyu Liu, Zhangyang Wang and Pan Li. 'Equivariant Hypergraph Diffusion Neural Operators'. In: The Eleventh International Conference on Learning Representations. 2022.

[73] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Peng Cui, P. Yu and Yanfang Ye. *Heterogeneous Graph Attention Network*. Comment: 10 pages. 2019. arXiv: 1903.07293 [cs.SI]. preprint.

[74] David S Wishart et al. 'DrugBank 5.0: a major update to the DrugBank database for 2018'. In: *Nucleic Acids Research* 46.D1 (2018), pp. D1074–D1082. ISSN: 0305-1048, 1362-4962.

[75] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie and Bin Cui. 'Graph Neural Networks in Recommender Systems: A Survey'. In: *ACM Comput. Surv.* 55.5 (2022), 97:1–97:37. ISSN: 0360-0300.

[76] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis and Partha Talukdar. 'HyperGCN: A New Method For Training Graph Convolutional Networks on Hypergraphs'. In: *Advances in Neural Information Processing Systems*. NeurIPS 2019. Vol. 32. Curran Associates, Inc., 2019.

[77] Muhammed A. Yildirim, Kwang-Il Goh, Michael E. Cusick, Albert-László Barabási and Marc Vidal. 'Drug-target network'. In: *Nat Biotechnol* 25.10 (2007), pp. 1119–1126. ISSN: 1087-0156. pmid: 17921997.

[78] Fouad El Zein and Jawad Snoussi. 'Local Systems and Constructible Sheaves'. In: *Arrangements, Local Systems and Singularities*. Ed. by Fouad El Zein, Alexandru I. Suciu, Meral Tosun, A. Muhammed Uludağ and Sergey Yuzvinsky. Basel: Birkhäuser, 2010, pp. 111–153. ISBN: 978-3-0346-0209-9.

[79] Xiangxiang Zeng et al. 'Target identification among known drugs by deep learning from heterogeneous networks'. In: *Chemical Science* 11.7 (2020), pp. 1775–1797. ISSN: 2041-6520.

[80] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami and Nitesh V. Chawla. 'Heterogeneous Graph Neural Network'. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 793–803. ISBN: 978-1-4503-6201-6.

Appendices

Appendix A

Chapter 5 Supplementary Material

A.1 Dataset information

A.1.1 Heterogeneous node classification

DBLP is a citation graph of bibliographic data for computer science with four node types: authors, papers, terms and venues. The aim is to predict which of the 4 classes an author node has.

IMDB is a graph of movie data and related metadata. The aim is to predict a movie's genre: action, comedy, drama, romance, and thriller. This is a multi-label classification task.

ACM is another graph of computer science citations. I use the subset from the original HAN paper [73]. The task is to classify the category of each paper.

A.1.2 Heterogeneous link prediction

LastFM [18] is a heterogeneous graph of music listening habits with three node types: user, artist and artist tags. The aim is to predict whether a user listens to an artist.

MovieLens[1] is a heterogeneous rating graph from the MovieLens website and consists of two node types: movie and user. The task is to predict whether or not a user recommends a movie.

A.1.3 Heterogeneous DTI prediction

DeepDTNet [79] consists of information about their drugs and their interaction between diseases and proteins from the Drug-Target Network [77], a bipartite graph of FDA approved drugs and proteins linked by interactions, and augmented by the DrugBank [43,

[1] https://movielens.org/

74] database. The data is provided as three incidence matrices: drug-protein interactions, drug-disease interactions, and protein-disease interactions. The task is to predict whether a particular drug will interact with the protein.

KEGG [51] consists of interactions between drugs, genes, and disease from the KEGG [40] biological database of genenomic data. Like with DeepDTNet, the raw data is three incidence matrices. The aim is to predict possible interactions between drugs and genes.

A.2 Restriction map implementation

Each restriction map is computed as $\mathcal{F}_{u \trianglelefteq e} = \text{MLP}(\mathbf{x}_u \,\|\, \mathbf{x}_v)$ in the case of graphs, and $\mathcal{F}_{u \trianglelefteq e} = \text{MLP}(\mathbf{x}_u \,\|\, \mathbf{h}_e)$ for hypergraphs.

Diagonal restriction maps. For the diagonal restriction maps, the MLP outputs d elements which are used to define the diagonal of the restriction map.

General restriction maps. For general restriction maps, the MLP outputs d^2 elements that are then rearranged to form a $d \times d$ matrix representing the restriction map.

Orthogonal restriction maps. For orthogonal restriction maps, MLP outputs $\left\lfloor \frac{d(d-1)}{2} \right\rfloor$ elements, and then the Torch Householder library [52] is used to generate a $d \times d$ orthogonal restriction map.

Low rank restriction maps. To predict a rank r restriction map, the MLP outputs $2dr + d$ elements that are rearranged into two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{d \times r}$ and a vector $\mathbf{c} \in \mathbb{R}^d$. The final restriction map is computed as $\mathbf{A}\mathbf{B}^\top + \text{diag}(\mathbf{c})$.

Appendix B

Chapter 6 supplementary material

B.1 Proofs

Theorem 6.1. *Neural Sheaf Diffusion is a special case of Sheaf-MPNN.*

Proof. Let $\tilde{\mathbf{X}} = D^{-1/2}(\mathbf{I}_n \otimes \mathbf{W}_1)\,\mathbf{X}\,\mathbf{W}_2$. Equation (2.11) can be expressed node-wise as

$$\mathbf{x}_u^{(k+1)} = \mathbf{x}_u^{(k)} - \sigma\left(D_u^{-1/2} \sum_{u,v \trianglelefteq e} \mathcal{F}_{u \trianglelefteq e}^{\top} (\mathcal{F}_{u \trianglelefteq e}\, \tilde{\mathbf{x}}_v - \mathcal{F}_{v \trianglelefteq e}\, \tilde{\mathbf{x}}_v) \right).$$

This is a Sheaf-MPNN defined as

$$\mathbf{m}_u^{(k+1)} = \sigma\left(D_u^{-1/2} \sum_{u,v \trianglelefteq e} \mathcal{F}_{u \trianglelefteq e}^{\top} \left(\mathcal{F}_{u \trianglelefteq e}\, \tilde{\mathbf{x}}_u^{(k)} - \mathcal{F}_{v \trianglelefteq e}\, \tilde{\mathbf{x}}_v \right) \right)$$

and update function $\mathbf{x}_u^{(k+1)} = \mathbf{x}_u^{(k)} - \mathbf{m}_u^{(k+1)}$. □

Theorem 6.2. *SheafAN and Res-SheafAN are special cases of Sheaf-MPNN.*

Proof. Let $\tilde{\mathbf{X}} = (\mathbf{I}_n \otimes \mathbf{W}_1)\,\mathbf{X}\,\mathbf{W}_2$. A SheafAN layer can be rewritten node-wise as

$$\mathbf{x}_u^{(k)} = \sigma\left(\sum_{v \in \mathcal{N}_u} \alpha_{uv} \mathbf{P}_{uv} \tilde{\mathbf{x}}_v^{(k)} \right)$$

where $\alpha_{uv} = a\left(\mathbf{x}_u^{(k)}, \mathbf{x}_v^{(k)}\right)$ and $\mathbf{P}_{uv} = \mathcal{F}_{u \trianglelefteq e}^{\top} \mathcal{F}_{v \trianglelefteq e}$. This is a Sheaf-MPNN with the message function $\mathbf{m}_u^{(k+1)} = \sum_{v \in \mathcal{N}_u} \alpha_{uv}\, \mathcal{F}_{u \trianglelefteq e}^{\top} \mathcal{F}_{v \trianglelefteq e}\, \mathbf{x}_v$ and the update function $\mathbf{x}_u^{(k+1)} = \mathbf{m}_u^{(k+1)}$. A Res-SheafAN layer can be expressed node-wise as

$$\mathbf{x}_u^{(k)} = \sigma\left(\sum_{v \in \mathcal{N}_u} \alpha_{uv} \mathbf{P}_{uv} \tilde{\mathbf{x}}_v^{(k)} - \tilde{\mathbf{x}}_u^{(k)} \right),$$

which defines a Sheaf-MPNN with the same message function as a SheafAN but with the update function $\mathbf{x}_u^{(k+1)} = \mathbf{m}_u^{(k+1)} - \tilde{\mathbf{x}}_u^{(k)}$. □

Theorem 6.3. *MPNNs are a special case of Sheaf-MPNN.*

Proof. Equation (2.1) may be recovered by using the trivial sheaf. □

Theorem 6.4. *SheafHyperGNN and SheafHyperGCN are special cases of SheafAllSet.*

Proof. Let $\tilde{\mathbf{X}} = D^{-1/2}(\mathbf{I}_n \otimes \mathbf{W}_2)\,\mathbf{X}\,\mathbf{W}_2$. First, I prove that a SheafHyperGNN layer is a special case of SheafAllSet. A SheafHyperGNN layer update can be expressed node-wise as

$$\mathbf{x}_u^{(k+1)} = \sigma\left(D_u^{-1/2}\tilde{\mathbf{x}}_u^{(k)} - D_u^{-1/2}\sum_{e;u\in e}\frac{1}{\delta_e}\mathcal{F}_{u\trianglelefteq e}^{\top}\left(\sum_{v\in e\setminus u}\left(\mathcal{F}_{u\trianglelefteq e}\tilde{\mathbf{x}}_u^{(k)} - \mathcal{F}_{u\trianglelefteq e}\tilde{\mathbf{x}}_v^{(k)}\right)\right)\right).$$

This is a SheafAllSet model such that

$$\mathbf{h}_e^{(k+1),u} = f_{V\to E}\left(V_{e\setminus u}^{(k)}; \mathbf{h}_e^{(k)}; \mathcal{F}_{u\trianglelefteq e}\,\mathbf{x}_u^{(k)}\right) = \sum_{v\in e\setminus u}\left(\mathcal{F}_{u\trianglelefteq e}\tilde{\mathbf{x}}_u^{(k)} - \mathcal{F}_{v\trianglelefteq e}\tilde{\mathbf{x}}_v^{(k)}\right),$$
$$\mathbf{x}_u^{(k+1)} = f_{E\to V}\left(E_u^{(k+1)}; \tilde{\mathbf{x}}_u^{(k)}\right) = \sigma\left(D_u^{-1/2}\tilde{\mathbf{x}}_u^{(k)} - D_u^{-1/2}\sum_{e;u\in e}\frac{1}{\delta_e}\mathcal{F}_{u\trianglelefteq e}^{\top}\,\mathbf{h}_e^{(k+1),u}\right).$$

The non-linear Laplacian used in the SheafHyperGCN layer may be reformulated as

$$\bar{L}_{\mathcal{F}}(\mathbf{x})_u := \sum_{e;u\in e}\frac{1}{\delta_e}\mathcal{F}_{u\trianglelefteq e}^{\top}\left(\sum_{v\in e} w_{uv,e}(\mathcal{F}_{u\trianglelefteq e}\,\mathbf{x}_u - \mathcal{F}_{v\trianglelefteq e}\,\mathbf{x}_v)\right)$$

where w_{uv} are weights defined as

$$w_{uv,e} = \begin{cases} \frac{1}{\delta_e} & \text{if } u \sim_e v \\ 0 & \text{otherwise} \end{cases}$$

The SheafHyperGCN update equation can be rewritten node-wise as

$$\mathbf{x}_u^{(k+1)} = \sigma\left(D_u^{-1/2}\tilde{\mathbf{x}}_u^{(k)} - D_u^{-1/2}\sum_{e;u\in e}\mathcal{F}_{u\trianglelefteq e}^{\top}\left(\sum_{v\in e} w_{uv,e}\left(\mathcal{F}_{u\trianglelefteq e}\tilde{\mathbf{x}}_u^{(k)} - \mathcal{F}_{v\trianglelefteq e}\tilde{\mathbf{x}}_v^{(k)}\right)\right)\right),$$

where $\tilde{\mathbf{x}}_u = \mathbf{W}_1\,\mathbf{x}_u\,\mathbf{W}_2$. This is a SheafAllSet model where $\tilde{\mathbf{x}}_u = D_u^{-1/2}\,\mathbf{W}_1\,\mathbf{x}_u\,\mathbf{W}_2$ and

$$\mathbf{h}_e^{(k+1),u} = f_{V\to E}\left(V_{e\setminus u}^{(k)}; \mathbf{h}_e^{(k)}; \mathcal{F}_{u\trianglelefteq e}\,\tilde{\mathbf{x}}_u^{(k)}\right) = \sum_{v\in e} w_{uv,e}\left(\mathcal{F}_{u\trianglelefteq e}\tilde{\mathbf{x}}_u^{(k)} - \mathcal{F}_{v\trianglelefteq e}\tilde{\mathbf{x}}_v^{(k)}\right),$$
$$\mathbf{x}_u^{(k+1)} = f_{E\to V}\left(E_u^{(k+1)}; \mathbf{x}_u^{(k)}\right) = \sigma\left(D_u^{-1/2}\tilde{\mathbf{x}}_u^{(k)} - D_u^{-1/2}\sum_{e;u\in e}\mathcal{F}_{u\trianglelefteq e}^{\top}\,\mathbf{h}_e^{(k+1),u}\right).$$

□

Theorem 6.5. *AllSet is a special case of SheafAllSet.*

Proof. AllSet may be recovered by using the trivial sheaf with SheafAllSet. □

Theorem 6.6. *Sheaf-MPNN is the special case of SheafAllSet applied to graphs.*

Proof. Equation (6.1) may be rewritten as the SheafAllSet update rule

$$\mathbf{h}_e^{(k+1),u} = f_{\mathcal{V}\to\mathcal{E}}\left(V_{e\setminus u}^{(k)}; \mathbf{h}_e^{(k)}\right) = f_{\mathcal{V}\to\mathcal{E}}\left(\left\{\mathcal{F}_{v\trianglelefteq e}\,\mathbf{x}_v^{(k)}\right\}; \mathbf{h}_e^{(k)}\right) = \mathcal{F}_{v\trianglelefteq e}\,\mathbf{x}_v^{(k)}$$
$$\mathbf{x}_u^{(k+1)} = f_{\mathcal{E}\to\mathcal{V}}\left(E_u^{(k+1)}; \mathbf{x}_u^{(k)}\right) = \varphi_u\left(\mathbf{x}_u^{(k)}, \bigoplus_{e:u\in e} \mathcal{F}_{u\trianglelefteq e}^{\top}\,\psi_e\left(\mathbf{h}_e^{(k+1),u}\right)\right).$$

□

Theorem 6.7. *A SheafSetTransformer is a universal approximator for Sheaf-MPNNs.*

Proof. By Theorem 1 of Lee et al. [44], a SheafSetTransformer layer can be expressed as

$$\mathrm{MLP}\left(\sum_{u,v\trianglelefteq e} \mathrm{MLP}\Big(\mathrm{Vec}\Big(\mathcal{F}_{u\trianglelefteq e}^{\top}\,\mathrm{MLP}\big(\mathcal{F}_{v\trianglelefteq e}\,\mathbf{x}_v\big)\Big)\Big)\right).$$

As matrix vectorisation is a linear operation, it may be accounted for as part of the MLP. This means that a SheafSetTransformer layer may be rewritten as

$$\mathrm{MLP}\left(\sum_{u,v\trianglelefteq e} \mathrm{MLP}\Big(\mathcal{F}_{u\trianglelefteq e}^{\top}\,\mathrm{MLP}\big(\mathcal{F}_{v\trianglelefteq e}\,\mathbf{x}_v\big)\Big)\right),$$

or as a SheafDeepSetLayer. As such, it must also universally approximate a Sheaf-MPNN. □

Theorem 6.8. *SheafDeepAllSets and SheafAllSetTransformer are universal approximators of SheafAllSet.*

Proof. By Tabaghi and Wang [64, Theorem 7], SheafDeepAllSets is a universal approximator of SheafAllSet. As a consequence of Proposition 4.1 of Chien et al. [10], a SheafAllSetTransformer transformer layer may be expressed as

$$f_{\mathcal{V}\to\mathcal{E}}\left(\tilde{V}_e^{(k)}\right) = \mathrm{MLP}\left(\sum_{u\in e} \mathrm{MLP}\big(\mathrm{Vec}\big(\mathcal{F}_{u\trianglelefteq e}\,\mathbf{x}_u\big)\big)\right)$$
$$f_{\mathcal{V}\to\mathcal{E}}\left(\tilde{E}_u^{(k+1)}\right) = \mathrm{MLP}\left(\sum_{u\in e} \mathrm{MLP}\Big(\mathrm{Vec}\Big(\mathcal{F}_{u\trianglelefteq e}^{\top}\,\mathbf{h}_e\Big)\Big)\right).$$

As matrix vectorisation is a linear operation, it may be universally approximated by an MLP. So, a SheafAllSetTransformer layer may be expressed as a SheafDeepAllSets layer and universally approximates SheafAllSet. □