SHEAF NEURAL NETWORKS WITH CONNECTION LAPLACIANS

Federico Barbero
University of Cambridge
fb548@cam.ac.uk

Cristian Bodnar
University of Cambridge
cb2015@cam.ac.uk

Haitz Sáez de Ocáriz Borde
University of Cambridge
hs788@cam.ac.uk

Michael Bronstein
University of Oxford & Twitter
mbronstein@twitter.com

Petar Veličković
DeepMind
petarv@google.com

Pietro Liò
University of Cambridge
pl219@cam.ac.uk

ABSTRACT

A Sheaf Neural Network (SNN) is a type of Graph Neural Network (GNN) that operates on a sheaf, an object that equips a graph with vector spaces over its nodes and edges and linear maps between these spaces. SNNs have been shown to have useful theoretical properties that help tackle issues arising from heterophily and over-smoothing. One complication intrinsic to these models is finding a good sheaf for the task to be solved. Previous works proposed two diametrically opposed approaches: manually constructing the sheaf based on domain knowledge and learning the sheaf end-to-end using gradient-based methods. However, domain knowledge is often insufficient, while learning a sheaf could lead to overfitting and significant computational overhead. In this work, we propose a novel way of computing sheaves drawing inspiration from Riemannian geometry: we leverage the manifold assumption to compute manifold-and-graph-aware orthogonal maps, which optimally align the tangent spaces of neighbouring data points. We show that this approach achieves promising results with less computational overhead when compared to previous SNN models. Overall, this work provides an interesting connection between algebraic topology and differential geometry, and we hope that it will spark future research in this direction.

1 INTRODUCTION

Graph Neural Networks (GNNs) (Scarselli et al., 2008) have shown encouraging results in a wide range of applications, ranging from drug design (Stokes et al., 2020) to guiding discoveries in pure mathematics (Davies et al., 2021). One advantage over traditional neural networks is that they can leverage the extra structure in graph data, such as edge connections.

GNNs, however, do not come without issues. Traditional GNN models, such as Graph Convolutional Networks (GCNs) (Kipf & Welling, 2016) have been shown to work poorly on heterophilic data. In fact, GCNs use homophily as an inductive bias by design, that is, they assume that connected nodes will likely belong to the same class and have similar feature vectors, which is not true in many real-world applications (Zhu et al., 2020a). Moreover, GNNs also suffer from over-smoothing (Oono & Suzuki, 2019), which prevents these models from improving, and may actually even worsen their performance when stacking several layers. These two problems are, from a geometric point of view, intimately connected (Chen et al., 2020a; Bodnar et al., 2022).

Bodnar et al. (2022) showed that when the underlying "geometry" of the graph is too simple, the issues discussed above arise. More precisely, they analysed the geometry of the graph through cellular sheaf theory (Curry, 2014; Hansen, 2020; Hansen & Ghrist, 2019), a subfield of algebraic topology (Hatcher, 2000). A cellular sheaf associates a vector space to each node and edge of a graph, and linear maps between these spaces. A GNN which operates over a cellular sheaf is known as a Sheaf Neural Network (SNN) (Hansen & Gebhart, 2020; Bodnar et al., 2022).

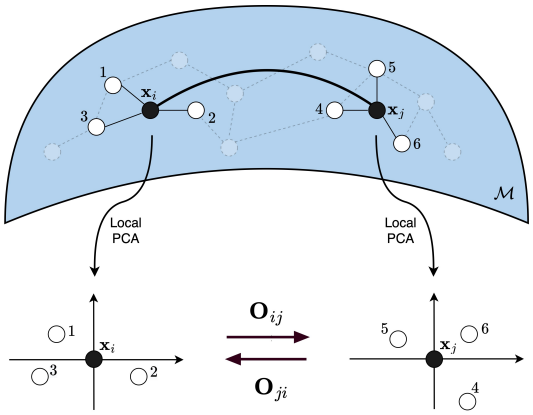

Figure 1: The orthonormal bases of $T_{\mathbf{x}_i}\mathcal{M}$ and $T_{\mathbf{x}_j}\mathcal{M}$ are determined by local PCA using nodes in the 1-hop neighbourhood of $\mathbf{x}_i$ and $\mathbf{x}_j$ respectively. The orthogonal mapping O_{ij} is a map from $T_{\mathbf{x}_i}\mathcal{M}$ to $T_{\mathbf{x}_j}\mathcal{M}$ which optimally aligns their bases.

SNNs work by computing a sheaf Laplacian, which recovers the well-known graph Laplacian when the underlying sheaf is trivial, that is, when vector spaces are 1-dimensional and we apply identity maps between them. Hansen & Ghrist (2019) have first shown the utility of SNNs in a toy experimental setting, where they used a manually-constructed sheaf Laplacian based on full knowledge of the data generation process. Bodnar et al. (2022) proposed to *learn* this sheaf Laplacian from data using stochastic gradient descent, making these types of models applicable to any graph dataset. However, this can also lead to computational complexity problems, overfitting and optimisation issues.

This work proposes a novel technique that aims to precompute a sheaf Laplacian from data in a deterministic manner, removing the need to learn it with gradient-based approaches. We do this through the lens of differential geometry, by assuming that the data is sampled from a low-dimensional manifold and optimally aligning the neighbouring tangent spaces via orthogonal transformations (see Figure 1). This idea was first introduced as groundwork for vector diffusion maps by Singer & Wu (2012). However, it only assumed a point-cloud structure. Instead, one of our contributions involves the computation of these optimal alignments over a graph structure. We find that our proposed technique performs well, while reducing the computational overhead involved in learning the sheaf.

In Section 2, we present a brief overview of cellular sheaf theory and neural sheaf diffusion (Bodnar et al., 2022). Next, in Section 3, we give details of our new procedure used to pre-compute the sheaf Laplacian before the model-training phase, which we refer to as Neural Sheaf Diffusion with Connection Laplacians (Conn-NSD). We then, in Section 4, evaluate this technique on various datasets with varying homophily levels. We believe that this work is a promising attempt at connecting ideas from algebraic topology and differential geometry with machine learning, and hope that it will spark further research at their intersection.

2 Background

We briefly overview the necessary background, starting with GNNs and cellular sheaf theory and concluding with neural sheaf diffusion. The curious reader may refer to Curry (2014); Hansen (2020); Hansen & Ghrist (2019) for a more in-depth insight into cellular sheaf theory, and to Bodnar et al. (2022) for the full theoretical results of neural sheaf diffusion.

2.1 Graph Neural Networks

GNNs are a family of neural network architectures that generalise neural networks to arbitrarily structured graphs. A graph $G = (V, E)$ is a tuple consisting of a set of nodes V and a set of edges E. We can represent each node in the graph with a d-dimensional feature vector $\mathbf{x}_v$ and group all the $n = |V|$ feature vectors into a $n \times d$ matrix $\mathbf{X}$. We represent the set of edges E with an adjacency

matrix $\mathbf{A}$. A GNN layer then takes these two matrices as input to produce a new set of (latent) feature vectors for each node:

$$\mathbf{H}^{(l)} = f\left(\mathbf{H}^{(l-1)}, \mathbf{A}\right). \tag{1}$$

In the case of a multi-layer GNN, the first layer $l = 1$, takes as input $\mathbf{H}^{(0)} = \mathbf{X}$, whereas subsequent layers, l, take as input $\mathbf{H}^{(l-1)}$, the latent features produced by the GNN layer immediately before it. There are numerous architectures which take this form, with one of the most popular being the Graph Convolutional Network (GCN) (Kipf & Welling, 2016) which implements Equation (1) the following way:

$$\mathbf{H}^{(l)} = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}\right), \tag{2}$$

where σ is a non-linear activation function (e.g. ReLU), $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\hat{\mathbf{D}}$ is the diagonal node degree matrix of $\hat{\mathbf{A}}$ and $\mathbf{W}^{(l)}$ is a weight matrix. This update propagation is local (due to the adjacency matrix), meaning that each latent feature vector is updated as a function of its local neighbourhood, weighted by a weight matrix and then symmetrically normalised. This kind of model has proven to be extremely powerful in a myriad of tasks. The weight matrix $\mathbf{W}^{(l)}$ at each layer is learnt from the data through back-propagation, by minimising some loss function (e.g. cross-entropy loss).

2.2 Cellular Sheaf Theory

Definition 2.1. A cellular sheaf $(G, \mathcal{F})$ on an *undirected graph* $G = (V, E)$ consists of:

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

The vector spaces of the node and edges are called *stalks*, while the linear maps are called *restriction maps*. It is then natural to group the various spaces. The space which is formed by the node stalks is called the space of 0-cochains, while the space formed by edge stalks is called the space of 1-cochains.

Definition 2.2. Given a sheaf $(G, \mathcal{F})$, we define the space of 0-cochains $C^0(G, \mathcal{F})$ as the direct sum over the vertex stalks $C^0(G, \mathcal{F}) := \bigoplus_{v \in V} \mathcal{F}(v)$. Similarly, the space of 1-cochains $C^1(G, \mathcal{F})$ as the direct sum over the edge stalks $C^1(G, \mathcal{F}) := \bigoplus_{e \in E} \mathcal{F}(e)$.

Defining the spaces $C^0(G, \mathcal{F})$ and $C^1(G, \mathcal{F})$ allows us to construct a linear *co-boundary map* $\delta : C^0(G, \mathcal{F}) \to C^1(G, \mathcal{F})$. From an opinion dynamics perspective (Hansen & Ghrist, 2021), the node stalks may be thought of as the private space of opinions and the edge stalks as the space in which these opinions are shared in a public discourse space. The co-boundary map δ then measures the disagreement between all the nodes.

Definition 2.3. Given some arbitrary orientation for each edge $e = u \to v \in E$, we define the co-boundary map $\delta : C^0(G, \mathcal{F}) \to C^1(G, \mathcal{F})$ as $\delta(\mathbf{x})_e = \mathcal{F}_{v \trianglelefteq e}\mathbf{x}_v - \mathcal{F}_{u \trianglelefteq e}\mathbf{x}_u$. Here $\mathbf{x} \in C^0(G, \mathcal{F})$ is a 0-cochain and $\mathbf{x}_v \in \mathcal{F}(v)$ is the vector of $\mathbf{x}$ at the node stalk $\mathcal{F}(v)$.

The co-boundary map δ allows us to construct the *sheaf Laplacian operator* over a sheaf.

Definition 2.4. The sheaf Laplacian of a sheaf is a map $L_{\mathcal{F}} : C^0(G, \mathcal{F}) \to C^0(G, \mathcal{F})$ defined as $L_{\mathcal{F}} = \delta^\top \delta$.

The sheaf Laplacian is a symmetric positive semi-definite (by construction) block matrix. The diagonal blocks are $L_{\mathcal{F}_{v,v}} = \sum_{v \trianglelefteq e} \mathcal{F}^\top_{v \trianglelefteq e}\mathcal{F}_{v \trianglelefteq e}$, while the off-diagonal blocks are $L_{\mathcal{F}_{v,u}} = -\mathcal{F}^\top_{v \trianglelefteq e}\mathcal{F}_{u \trianglelefteq e}$.

Definition 2.5. The *normalised sheaf Laplacian* $\Delta_{\mathcal{F}}$ is defined as $\Delta_{\mathcal{F}} = D^{-\frac{1}{2}} L_{\mathcal{F}} D^{-\frac{1}{2}}$ where D is the block-diagonal of $L_{\mathcal{F}}$.

Although stalk dimensions are arbitrary, we work with node and edge stalks which are all d-dimensional for simplicity. This means that each restriction map is $d \times d$, and therefore so is each block in the sheaf Laplacian. With n we denote the number of nodes in the underlying graph G, which results in our sheaf Laplacian having dimensions $nd \times nd$.

If we construct a trivial sheaf where each stalk is isomorphic to $\mathbb{R}$ and the restriction maps are identity maps, then we recover the well-known $n \times n$ graph Laplacian from the sheaf Laplacian. This effectively means that the sheaf Laplacian generalises the graph Laplacian by considering a non-trivial sheaf on G.

Definition 2.6. The *orthogonal (Lie) group* of dimension d, denoted $O(d)$, is the group of $d \times d$ orthogonal matrices together with matrix multiplication.

If we constrain the restriction maps in the sheaf to belong to the orthogonal group (i.e., $\mathcal{F}_{v \trianglelefteq e} \in O(d)$), the sheaf becomes a *discrete* $O(d)$*-bundle* and can be thought of as a discretised version of a tangent bundle on a manifold. The sheaf Laplacian of the $O(d)$-bundle is equivalent to a *connection Laplacian* used by Singer & Wu (2012). The orthogonal restriction maps describe how vectors are rotated when transported between stalks, in a way analogous to the transportation of tangent vectors on a manifold.

Orthogonal restriction maps are advantageous because orthogonal matrices have fewer free parameters, making them more efficient to work with. The Lie group $O(d)$ has a $d(d-1)/2$-dimensional manifold structure (compared to the d^2-dimensional general linear group describing all invertible matrices). In $d = 2$, for instance, 2×2 rotation matrices have only one free parameter (the rotation angle).

2.3 Neural Sheaf Diffusion

We now discuss the existing sheaf-based machine learning models and their theoretical properties. Consider a graph $G = (V, E)$ where each node $v \in V$ has a d-dimensional feature vector $\mathbf{x}_v \in \mathcal{F}(v)$. We construct an nd-dimensional vector $\mathbf{x} \in C^0(G, \mathcal{F})$ by column-stacking the individual vectors $\mathbf{x}_v$. Allowing for f feature channels, we produce the feature matrix $\mathbf{X} \in \mathbb{R}^{(nd) \times f}$. The columns of $\mathbf{X}$ are vectors in $C^0(G, \mathcal{F})$, one for each of the f channels.

Sheaf diffusion is a process on $(G, \mathcal{F})$ governed by the following differential equation:

$$\mathbf{X}(0) = \mathbf{X}, \quad \dot{\mathbf{X}}(t) = -\Delta_{\mathcal{F}} \mathbf{X}(t), \tag{3}$$

which is discretised via the explicit Euler scheme with unit step-size:

$$\mathbf{X}(t+1) = \mathbf{X}(t) - \Delta_{\mathcal{F}} \mathbf{X}(t) = \left(\mathbf{I}_{nd} - \Delta_{\mathcal{F}}\right) \mathbf{X}(t)$$

The model used by Bodnar et al. (2022) for experimental validation was of the form

$$\dot{\mathbf{X}} = -\sigma\left(\Delta_{\mathcal{F}(t)} \left(\mathbf{I}_n \otimes \mathbf{W}_1\right) \mathbf{X}(t) \mathbf{W}_2\right) \tag{4}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices, the restriction maps defining $\Delta_{\mathcal{F}(t)}$ are computed by a learnable parametric matrix-valued function $\mathcal{F}_{v \trianglelefteq e:=(v,u)} = \mathbf{\Phi}(\mathbf{x}_v, \mathbf{x}_u)$, on which additional constraints (e.g., diagonal or orthogonal structure) can be imposed. Equation (4) was discretised as

$$\mathbf{X}_{t+1} = \mathbf{X}_t - \sigma\left(\Delta_{\mathcal{F}(t)} \left(\mathbf{I}_n \otimes \mathbf{W}_1^t\right) \mathbf{X}_t \mathbf{W}_2^t\right) \tag{5}$$

It is important to note that the sheaf $\mathcal{F}(t)$ and the weights $\mathbf{W}_1^t, \mathbf{W}_2^t$ in equation (5) are time-dependent, meaning that the underlying "geometry" evolves over time.

3 Connection Sheaf Laplacians

The sheaf Laplacian $\Delta_{\mathcal{F}(t)}$ arises from the sheaf $\mathcal{F}(t)$ built upon the graph G, which in turn is determined by constructing the individual restriction maps $\mathcal{F}_{v \trianglelefteq e}$. Instead of learning a parametric function $\mathcal{F}_{v \trianglelefteq e:=(v,u)} = \mathbf{\Phi}(\mathbf{x}_v, \mathbf{x}_u)$ as done by Bodnar et al. (2022), we compute the restriction maps in a non-parametric manner at pre-processing time. In doing so, we avoid learning the maps by backpropagation. In particular the restriction maps we compute are orthogonal. We work with this class because it was shown to be more efficient when using the same stalk width as compared to other models in Bodnar et al. (2022), and due to the geometric analogy to parallel transport on manifolds.

3.1 Local PCA & Alignment for Point Clouds

We adapt a procedure to learn orthogonal transformations on point clouds, presented by Singer & Wu (2012). Their construction relies on the so-called "manifold assumption", positing that even though

data lives in a high-dimensional space $\mathbb{R}^p$, the correlation between dimensions suggests that in reality, the data points lie on a d-dimensional Riemannian manifold $\mathcal{M}^d$ embedded in $\mathbb{R}^p$ (with significantly lower dimension, $d \ll p$).

Assume the manifold $\mathcal{M}^d$ is sampled at points $\{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^p$. At every point $\mathbf{x}_i$, $\mathcal{M}^d$ has a *tangent space* $T_{\mathbf{x}_i}\mathcal{M}$ (which is analogous to our $\mathcal{F}(v)$) that intuitively contains all the vectors at $\mathbf{x}_i$ that are tangent to the manifold. A mechanism allowing to transport vectors between two $T_{\mathbf{x}_i}\mathcal{M}$ and $T_{\mathbf{x}_j}\mathcal{M}$ at nearby points is a *connection* (or *parallel transport*, which would correspond to our transport maps $\mathcal{F}_{v \trianglelefteq e}^{\top}\mathcal{F}_{u \trianglelefteq e}$ between $\mathcal{F}(u)$ and $\mathcal{F}(v)$).

Computing a connection on the discretised manifold is a two step procedure. First, orthonormal bases of the tangent spaces for each data point are constructed via local PCA. Next, the tangent spaces are optimally aligned via orthogonal transformations, which can be thought of as mappings from one tangent space to a neighbouring one. Singer & Wu (2012) computed a $\sqrt{\epsilon_{PCA}}$-neighbourhood ball of points for each point $\mathbf{x}_i$ denoted $\mathcal{N}_{\mathbf{x}_i, \epsilon_{PCA}}$. This forms a set of neighbouring points $\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_{N_i}}$. Then the $p \times N_i$ matrix $\hat{\mathbf{X}}_i = [\mathbf{x}_{i_1} - \mathbf{x}_i, \dots, \mathbf{x}_{i_{N_i}} - \mathbf{x}_i]$ is obtained, which centres all of the neighbours at $\mathbf{x}_i$. Next, an $N_i \times N_i$ weighting matrix $\mathbf{D}_i$ is constructed, giving more importance to neighbours closer to $\mathbf{x}_i$. This allows us to compute the $p \times N_i$ matrix $\mathbf{B}_i = \hat{\mathbf{X}}_i\mathbf{D}_i$. Then Singular Value Decomposition (SVD) is used on $\mathbf{B}_i$ such that $\mathbf{B}_i = \mathbf{U}_i\mathbf{\Sigma}_i\mathbf{V}_i^{\top}$. Assuming that the singular values are in decreasing order, the first d left singular vectors are kept (the first d vectors of $\mathbf{U}_i$), forming the matrix $\mathbf{O}_i$. Note that the columns of $\mathbf{O}_i$ are orthonormal by construction and they form a d-dimensional subspace of $\mathbb{R}^p$. This basis constitutes our approximation to the basis of the tangent space $T_{\mathbf{x}_i}\mathcal{M}$.

To compute the orthogonal matrix $\mathbf{O}_{ij}$, which represents our orthogonal transformation from $T_{\mathbf{x}_i}\mathcal{M}$ to $T_{\mathbf{x}_j}\mathcal{M}$, it is sufficient to first of all compute the SVD of $\mathbf{O}_i^{\top}\mathbf{O}_j = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}$ and then $\mathbf{O}_{ij} = \mathbf{U}\mathbf{V}^{\top}$. $\mathbf{O}_{ij}$ is the orthogonal transformation which optimally aligns the tangent spaces $T_{\mathbf{x}_i}\mathcal{M}$ and $T_{\mathbf{x}_j}\mathcal{M}$ based on their bases $\mathbf{O}_i$ and $\mathbf{O}_j$. Whenever $\mathbf{x}_i$ and $\mathbf{x}_j$ are "nearby", Singer & Wu (2012) show that $\mathbf{O}_{ij}$ is an approximation to the parallel transport operator.

3.2 Local PCA & Alignment for Graphs

The technique has many valuable theoretical properties, but was originally designed for point clouds. In our case, we also wish to leverage the valuable edge information at our disposal. To do this, instead of computing the neighbourhood $\mathcal{N}_{\mathbf{x}_i, \epsilon_{PCA}}$, we take the 1-hop neighbourhood $\mathcal{N}_{\mathbf{x}_i}^1$ of $\mathbf{x}_i$. A problem is encountered when computing the weighting matrix $\mathbf{D}_i$, which gives different weightings dependent on the distance to the centroid of the neighbourhood. We make the assumption that $\mathbf{D}_i$ is an identity matrix, giving the same weighting to each node in the neighbourhood, as they are all at a 1-hop distance from the reference feature vector. This means that in our approach $\mathbf{B}_i = \hat{\mathbf{X}}_i\mathbf{D}_i = \hat{\mathbf{X}}_i$.

Following this modification, the technique matches the procedure proposed by Singer & Wu (2012). We compute the SVD of $\mathbf{B}_i$ to extract $\mathbf{O}_i$ from the left singular vectors. We finally compute the orthogonal transport maps $\mathbf{O}_{ij}$ from the SVD of $\mathbf{O}_i^{\top}\mathbf{O}_j$. This gives a modified version of the alignment procedure, that is now graph-aware. To the best of our knowledge, this a novel technique to operate over graphs. A diagram of the newly proposed approach is displayed in Figure 1.

Estimating d is non-trivial, that is, the dimension of the tangent space (in our case, the stalks). In fact, we are assuming that every neighbourhood is larger than d or else $\mathbf{B}_i$ would have less than d singular vectors, and our construction would be ill-defined. This is clearly not always the case for all d. While Singer & Wu (2012) proposed to estimate d directly from the data, we leave d as a tunable hyper-parameter.

To solve the problem for nodes which have less than d neighbours, we take the closest neighbours in terms of the Euclidean distance which are not in the 1-hop neighbourhood. In other words, when there are less than d neighbours, we pick the remaining neighbours following the original procedure by Singer & Wu (2012). We note that one could try to consider an n-hop neighbourhood instead, in a similar fashion to ϵ_{PCA} in the original technique. Still, this comes with a larger computational overhead and complications related to the weightings. Furthermore, if a graph has a disconnected node, this would still be an issue. In practice, d is kept small such that most nodes have at least d edge-neighbours.

Algorithm 1 Local PCA & Alignment for Graphs

Input: feature matrix $\mathbf{X}$, EdgeIndex, stalk dimension d
// Graph Local PCA
for $i = 0$ **to** len($\mathbf{X}$) **do**
  // 1-hop neighbourhood and closest vectors
  // (Euclidean distance) if needed, centred at x_i
  $\hat{\mathbf{X}}_i = \text{LocalNeighbourhood}(\mathbf{X}, \text{EdgeIndex}, \text{i})$
  $\mathbf{U}_i, \mathbf{\Sigma}_i, \mathbf{V}_i^\top = \text{SVD}(\hat{\mathbf{X}}_i)$
  // Choose first d left singular vectors
  $\mathbf{O}_i = \mathbf{U}_i[:, :d]$
end for
// Alignment
for i, j **in** EdgeIndex **do**
  $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^\top = \text{SVD}(\mathbf{O}_i^\top \mathbf{O}_j)$
  $\mathbf{O}_{ij} = \mathbf{U}\mathbf{V}^\top$
end for

Algorithm 1 shows the pseudo-code for our technique. In principle, the LocalNeighbourhood function selects the neighbours based on the 1-hop neighbourhood. If the number of these neighbours is less than the stalk dimension, we pick the closest neighbours based on the Euclidean distance, which are not in the 1-hop neighbourhood. Assuming unit cost for SVD, the run-time increases linearly with the number of data-points. Also, given that the approach here described is performed at pre-processing time, we are able to compute the sheaf Laplacian in a deterministic way in constant time during training. This removes the overhead required whilst backpropagating through the sheaf Laplacian to learn the parametric function $\mathbf{\Phi}$. It also helps counter issues related to overfitting, especially when the dimension of the stalks increases as we are removing the additional parameters which come with $\mathbf{\Phi}$, reducing model complexity.

4 Evaluation

We evaluate our model on several datasets, and compare its performance to a variety of models recorded in the literature, as well as to some especially designed baselines. For consistency, we use the same datasets as the ones discussed by Bodnar et al. (2022). These are real-world datasets which aim at evaluating heterophilic learning (Rozemberczki et al., 2021; Pei et al., 2020). They are ordered based on their homophily coefficient $0 \leq h \leq 1$, which is higher for more homophilic datasets. Effectively, h is the fraction of edges which connect nodes of the same class label. The results are collected over 10 fixed splits, where 48%, 32%, and 20% of nodes per class are used for training, validation, and testing, respectively. The reported results are chosen from the highest validation score.

Table 1 contains accuracy results for a wide range of models, along with ours, Conn-NSD, for node classification tasks. An important baseline is the Multi-Layer Perceptron (MLP), whose result we report in the last row of Table 1. The MLP has access only to the node features and it provides an idea of how much useful information GNNs can extract from the graph structure. The GNN models in Table 1 can be classified in 3 main categories:

1. Classical: GCN (Kipf & Welling, 2016), GAT (Velickovic et al., 2017), GraphSAGE (Hamilton et al., 2017),
2. Models for heterophilic settings: GGCN (Yan et al., 2021), Geom-GCN (Pei et al., 2020), H2GCN (Zhu et al., 2020b), GPRGNN (Chien et al., 2020), FAGCN (Bo et al., 2021), MixHop (Abu-El-Haija et al., 2019),
3. Models which address over-smoothing: GCNII (Chen et al., 2020b), PairNorm (Zhao & Akoglu, 2019),

Additionally, we also include the results presented by Bodnar et al. (2022) using sheaf diffusion models, and the two random baselines: RandEdge-NSD and RandNode-NSD. RandEdge-NSD generates the sheaf by sampling a Haar-random matrix (Meckes, 2019) for each edge. RandNode-NSD instead generates the sheaf by sampling a Haar-random matrix for each node $\mathbf{O}_i$ and then by

Table 1: Accuracy $\pm$ variance for various node classification datasets and models. The datasets are sorted by increasing order of homophily. Our technique is denoted Conn-NSD, while the other Sheaf Diffusion models are Diag-NSD, O(d)-NSD and Gen-NSD. The top three models are coloured by **First**, **Second** and **Third**, respectively. The first section includes sheaf-based models, while the second includes other GNN models.

	Texas	**Wisconsin**	**Film**	**Squirrel**	**Chameleon**	**Cornell**	**Citeseer**	**Pubmed**	**Cora**
Homophily level	**0.11**	**0.21**	**0.22**	**0.22**	**0.23**	**0.30**	**0.74**	**0.80**	**0.81**
#Nodes	183	251	7,600	5,201	2,277	183	3,327	18,717	2,708
#Edges	295	466	26,752	198,493	31,421	280	4,676	44,327	5,278
#Classes	5	5	5	5	5	5	7	3	6
Conn-NSD (ours)	**86.16**±2.24	**88.73**±4.47	**37.91**±1.28	45.19±1.57	65.21±2.04	**85.95**±7.72	75.61±1.93	89.28±0.38	83.74±2.19
RandEdge-NSD	84.05±5.33	85.69±4.02	37.40±1.18	33.89±1.56	47.72±1.60	84.59±7.65	72.49±1.91	87.74±0.50	74.00±1.99
RandNode-NSD	82.97±7.55	86.47±4.51	37.54±1.32	34.00±1.43	50.68±2.48	83.78±7.81	73.89±1.94	89.13±0.59	80.90±1.51
Diag-NSD	**85.67**±6.95	88.63±2.75	37.79±1.01	**54.78**±1.81	**68.68**±1.73	**86.49**±7.35	**77.14**±1.85	89.42±0.43	87.14±1.06
$O(d)$-NSD	**85.95**±5.51	**89.41**±4.74	**37.81**±1.15	**56.34**±1.32	**68.04**±1.58	84.86±4.71	76.70±1.57	**89.49**±0.40	86.90±1.13
Gen-NSD	82.97±5.13	**89.21**±3.84	**37.80**±1.22	53.17±1.31	67.93±1.58	**85.68**±6.51	76.32±1.65	89.33±0.35	87.30±1.15
GGCN	84.86±4.55	86.86±3.29	37.54±1.56	**55.17**±1.58	**71.14**±1.84	**85.68**±6.63	**77.14**±1.45	89.15±0.37	**87.95**±1.05
H2GCN	84.86±7.23	87.65±4.98	35.70±1.00	36.48±1.86	60.11±2.15	82.70±5.28	77.11±1.57	**89.49**±0.38	**87.87**±1.20
GPRGNN	78.38±4.36	82.94±4.21	34.63±1.22	31.61±1.24	46.58±1.71	80.27±8.11	77.13±1.67	87.54±0.38	**87.95**±1.18
FAGCN	82.43±6.89	82.94±7.95	34.87±1.25	42.59±0.79	55.22±3.19	79.19±9.79	N/A	N/A	N/A
MixHop	77.84±7.73	75.88±4.90	32.22±2.34	43.80±1.48	60.50±2.53	73.51±6.34	76.26±1.33	85.31±0.61	87.61±0.85
GCNII	77.57±3.83	80.39±3.40	37.44±1.30	38.47±1.58	63.86±3.04	77.86±3.79	**77.33**±1.48	**90.15**±0.43	**88.37**±1.25
Geom-GCN	66.76±2.72	64.51±3.66	31.59±1.15	38.15±0.92	60.00±2.81	60.54±3.67	**78.02**±1.15	**89.95**±0.47	85.35±1.57
PairNorm	60.27±4.34	48.43±6.14	27.40±1.24	50.44±2.04	62.74±2.82	58.92±3.15	73.59±1.47	87.53±0.44	85.79±1.01
GraphSAGE	82.43±6.14	81.18±5.56	34.23±0.99	41.61±0.74	58.73±1.68	75.95±5.01	76.04±1.30	88.45±0.50	86.90±1.04
GCN	55.14±5.16	51.76±3.06	27.32±1.10	53.43±2.01	64.82±2.24	60.54±5.30	76.50±1.36	88.42±0.50	86.98±1.27
GAT	52.16±6.63	49.41±4.09	27.44±0.89	40.72±1.55	60.26±2.50	61.89±5.05	76.55±1.23	87.30±1.10	86.33±0.48
MLP	80.81±4.75	85.29±3.31	36.53±0.70	28.77±1.56	46.21±2.99	81.89±6.40	74.02±1.90	75.69±2.00	87.16±0.37

computing the transport maps $\mathbf{O}_{ij}$ from $\mathbf{O}_i$ and $\mathbf{O}_j$. These last two baselines help us determine how our sheaf structure performs against a randomly sampled one.

As we can see from the results, sheaf diffusion models tend to perform best for the heterophilic datasets such as Texas, Wisconsin, and Film. On the other hand, their relative performance drops as homophily increases. This is expected since, for example, classical models such as GCN and GAT exploit homophily by construction, whereas sheaf diffusion models are more general, adaptable, and versatile, but at the same time lose the inductive bias provided by classical models for homophilic data.

Conn-NSD, alongside the other original discrete sheaf diffusion methods, consistently beats the random orthogonal sheaf baselines, which shows that our model incorporates meaningful geometric structure. The proposed Conn-NSD model achieves excellent results on the Texas and Film datasets, outperforming Diag-NSD, O(d)-NSD, and Gen-NSD, using fewer learnable parameters. Furthermore, Conn-NSD also obtains competitive results for Wisconsin, Cornell and Pubmed and remains close-behind on Citeseer and Cora.

It is only in the case of the Squirrel dataset, and to a lesser extent Chameleon, that Conn-NSD is not able to perform as well as the models discussed by Bodnar et al. (2022). The Squirrel dataset contains a large amount of nodes and a substantially greater number of edges than all the other datasets. Importantly, the underlying MLP used for classification scores poorly. It may be that the extra flexibility provided by learning the sheaf is specially beneficial in cases in which the underlying MLP achieves low accuracy. Nevertheless, Conn-NSD still convincingly outperforms the random baselines, especially on these last two datasets.

Overall, Conn-NSD performs comparably well to learning the sheaf via gradient-based approaches in most cases. It also seems most well-suited on graphs with a very low amount of nodes. This may be explained by the fact that Conn-NSD aims to mitigate overfitting, acting as a form of regularisation which allows for faster training and fewer parameters.

Runtime performance Finally, we measure the speedup achieved by moving the computation of the sheaf Laplacian at pre-processing time. Table 2 displays the mean wall-clock time for an epoch measured in seconds, obtained with a NVIDIA TITAN X GPU and an Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz. Conn-NSD achieves significantly faster inference times when compared to its direct counter-part $O(d)$-NSD from Bodnar et al. (2022). The larger datasets see the most benefit, with Squirrel showing a 45.8% speed up.

Table 2: Mean seconds per epoch for each of the datasets. The proposed model achieves faster inference times because it does not need to learn and build a Laplacian at each layer.

	Texas	**Wisconsin**	**Film**	**Squirrel**	**Chameleon**	**Cornell**	**Citeseer**	**Pubmed**	**Cora**
#Nodes	183	251	7,600	5,201	2,277	183	3,327	18,717	2,708
#Edges	295	466	26,752	198,493	31,421	280	4,676	44,327	5,278
Conn-NSD (ours)	0.010	0.013	0.017	0.310	0.169	0.013	0.011	0.147	0.015
$O(d)$-NSD	0.017	0.018	0.022	0.572	0.296	0.019	0.017	0.263	0.022

5 Conclusion

We proposed and evaluated a novel technique to compute the sheaf Laplacian of a graph deterministically, obtaining promising results. This was done by leveraging existing differential geometry work that constructs orthogonal maps that optimally align tangent spaces between points, relying on the manifold assumption. We crucially adapted this intuition to be graph-aware, leveraging the valuable edge connection information in the graph structure.

We showed that this technique achieves competitive empirical results and it is able to beat or match the performance of the original models by Bodnar et al. (2022) on most datasets, as well as to consistently outperform the random sheaf baselines. This suggests that in some cases it may not be necessary to learn the sheaf through a parametric function, but instead the sheaf can be computed as a pre-processing step. This work may be regarded as a regularisation technique for SNNs, which also reduces the training time as it removes the need to backpropagate through the sheaf.

We believe we have uncovered an exciting research direction which aims to find a way to compute sheaves non-parametrically with an objective that is independent of the downstream task. Furthermore, we are excited by the prospect of further research tying intuition stemming from the fields of algebraic topology and differential geometry to machine learning. We believe that this work forms a promising first step in this direction.

References

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pp. 21–29. PMLR, 2019.

Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. *arXiv preprint arXiv:2101.00797*, 2021.

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 *ICLR 2022 Workshop on Geometrical and Topological Representation Learning*, 2022.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3438–3445, 2020a.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pp. 1725–1735. PMLR, 2020b.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Joint adaptive feature smoothing and topology extraction via generalized pagerank gnns. *arXiv preprint arXiv:2006.07988*, 2020.

Justin Michael Curry. *Sheaves, cosheaves and applications*. University of Pennsylvania, 2014.

Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, et al. Advancing mathematics by guiding human intuition with ai. *Nature*, 600(7887):70–74, 2021.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Jakob Hansen. *Laplacians of Cellular Sheaves: Theory and Applications*. PhD thesis, University of Pennsylvania, 2020.

Jakob Hansen and Thomas Gebhart. Sheaf neural networks. *arXiv preprint arXiv:2012.06333*, 2020.

Jakob Hansen and Robert Ghrist. Toward a spectral theory of cellular sheaves. *Journal of Applied and Computational Topology*, 3(4):315–358, 2019.

Jakob Hansen and Robert Ghrist. Opinion dynamics on discourse sheaves. *SIAM Journal on Applied Mathematics*, 81(5):2033–2060, 2021.

Allen Hatcher. *Algebraic topology*. Cambridge Univ. Press, Cambridge, 2000.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Elizabeth S Meckes. *The random matrix theory of the classical compact groups*, volume 218. Cambridge University Press, 2019.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Amit Singer and H-T Wu. Vector diffusion maps and the connection laplacian. *Communications on pure and applied mathematics*, 65(8):1067–1144, 2012.

Jonathan M. Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M. Donghia, Craig R MacNair, Shawn French, Lindsey A. Carfrae, Zohar Bloom-Ackermann, Victoria M. Tran, Anush Chiappino-Pepe, Ahmed H. Badran, Ian W. Andrews, Emma J. Chory, George M. Church, Eric D. Brown, T. Jaakkola, Regina Barzilay, and James J. Collins. A deep learning approach to antibiotic discovery. *Cell*, 180:688–702.e13, 2020.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020a.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Generalizing graph neural networks beyond homophily. *arXiv preprint arXiv:2006.11468*, 2020b.