Neural Sheaf Diffusion for Graph Classification

Olga Zaghen 224436
University of Trento
olga.zaghen@studenti.unitn.it

Abstract

This work was carried out as final project for the course *Advanced Topics in Machine Learning and Optimization*, with the supervision of PhD student Antonio Longa. The primary goal of this study was understanding whether the benefits that sheaf diffusion networks bring to node classification tasks can be someway exploited in a different context such as graph classification.

1 Introduction

Graph Neural Networks (GNNs) have shown astonishing results in broad range of both practical and theorical applications, from drug design to guiding discoveries in mathematics. The main tasks that GNNs aim to solve in such variety of contexts mainly consist in node classification, graph classification and link prediction. The main advantage over traditional neural networks is their ability to leverage the complex structure that characterizes graph data.

Despite all the advantages they bring, GNNs also come with some drawbacks, such as the inability to perform satisfactorily in heteroliphic settings, due to the fact that traditional GNNs such as GCNs use homophily as inductive bias by design. This problem is connected to a further issue, that is over-smoothing, which may even worsen the performance of a GNN when stacking several layers.

It has been shown by Bodnar et al. (1) that those issues may arise when the graph's underlying geometry is too simple. For this reason they proposed to augment the expressive power of a graph and improve the information diffusion process between nodes by making the GNN operate over cellular sheaves, associating a non-trivial vector space to each node and edge and defining linear maps between them. These particular GNNs are known as Sheaf Neural Networks (SNNs) (1) (2).

The benefits provided by SNNs over *scalar* GNNs are particularly relevant when dealing with node classification tasks, especially in the case of heteroliphic data. One interesting question is whether their ability to overcome over-smoothing could be someway useful also in different settings, like for graph classification. To this end, the Neural Sheaf Diffusion (1) ideas and code [1] were adapted, testing different readout and pooling methods. Beyond this kind of extension, I noticed that the original algorithm is not designed to handle edge features, although they may be very important to consider for solving graph classification problems. Motivated by this, I came up with different ways to make edge features contribute to the sheaf diffusion process, and tried them out in practice to test their effect on the overall performance.

A Jupyter Notebook has been developed to display the experiments and the obtained results [2].

2 Background

In this section a brief overview of the necessary background is presented, covering some essential concepts about GNNs, cellular sheaf theory and neural sheaf diffusion. For a deeper insight on the latter mentioned topics, some useful resources are the works by Curry (3) and Hansen (2) for what concerns cellular sheaf theory, and Bodnar et al. (1) for neural sheaf diffusion.

2.1 Graph Neural Networks

GNNs are a set of models that generalize neural networks to graphs of arbitrary structure. A graph can be defined as a tuple $G = (V, E)$, V being its set of nodes and E being its set of edges. If we suppose $|V| = n$ and we associate to each node v an f-dimensional feature vector $\mathbf{x}_v$, we can think of grouping all feature vectors in a $n \times f$ matrix $\mathbf{X}$. The edge-level information can be also expressed in a compact way through the adjacency matrix $\mathbf{A}$. Each GNN layer (there may be more than one in multi-layer GNNs) processes these matrices to produce, for each node, a new set of updated feature vectors:

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

where l stands for the layer index, and the first layer takes as input $\mathbf{H}^{(0)} = \mathbf{X}$. Several GNN models are specific instances of the general form above, one of them being the Graph Convolutional Network (GCN) by Kipf and Welling (4) that adapts (1) as:

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

In this equation, σ is a non-linear activation function, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\hat{\mathbf{D}}$ is the diagonal node degree matrix of $\hat{\mathbf{A}}$, and $\mathbf{W}^{(l)}$ is the l-th layer weight matrix, learnt from data through back-propagation. Due to the presence of the adjacency matrix, this kind of update process is local: the update of a node's feature vector at each step depends only on its neighbors.

[1] https://github.com/twitter-research/neural-sheaf-diffusion

[2] https://github.com/olgatticus/NSD_GraphClassification

2.1.1 Graph Classification Task

The aforementioned model (2) has proven its effectiveness for a range of different tasks, among which graph classification. In graph classification problems the main goal is classifying entire graphs within a finite number of categories. GNNs that are specifically designed for this task aim at computing a meaningful graph-level embedding though permutation-invariant operations, in a way that the representations belonging to different categories become linearly separable in the embedding space. This is generally done by first extracting useful node-level embeddings and then combining them through a readout operation, or by gradually reducing the graph size stacking multiple pooling layers sequentially.

2.2 Cellular Sheaf Theory

Given an *undirected* graph $G = (V, E)$, a *cellular sheaf* F over G is composed of:

1. an assignment of a vector space $F(v)$ for each $v \in V$,
2. an assignment of a vector space $F(e)$ for each $e \in E$,
3. a linear map $F_{v \trianglelefteq e} : F(v) \to F(e)$ whenever v is adjacent to the edge e.

The vector spaces $F(v)$ and $F(e)$ are referred to as *stalks*, while the linear maps $F_{v \trianglelefteq e}$ are the *restriction maps*. In our application, elements of a vertex stalk $F(v)$ correspond to node-wise feature vectors $\mathbf{x}_v$, while the edge stalks $F(e)$ only serve as auxiliary spaces for mixing node features.

Given a sheaf $(G; F)$, one can define the space of 0-cochains $C^0(G; F)$ as the direct sum over the vertex stalks $C^0(G; F) := \oplus_{v \in V} F(v)$. The space of 1-cochains $C^1(G; F)$ is instead the direct sum over the edge stalks $C^1(G; F) := \oplus_{e \in E} F(e)$. The space 0-cochains roughly consists of all possible collections of feature vectors $\mathbf{x} = (\mathbf{x}_v)_{v \in V}$.

The *sheaf Laplacian* is a linear map $L_F : C^0(G; F) \to C^0(G; F)$ defined on each stalk as

$$L_F(\mathbf{x})_v = \sum_{u, v \trianglelefteq e} F^T_{v \trianglelefteq e}(F_{v \trianglelefteq e}\mathbf{x}_v - F_{u \trianglelefteq e}\mathbf{x}_u)$$

The sheaf Laplacian is a positive semidefinite block matrix, of which the diagonal blocks are $L_{F_{v,v}} = \sum_{v \trianglelefteq e} F^T_{v \trianglelefteq e} F_{v \trianglelefteq e}$ while the off-diagonal blocks are $L_{F_{v,u}} = -F^T_{v \trianglelefteq e} F_{u \trianglelefteq e}$. The *normalized sheaf Laplacian* Δ_F is defined as $\Delta_F = D^{-\frac{1}{2}} L_F D^{-\frac{1}{2}}$ where D is the block-diagonal of L_F.

For simplicity, we set the dimension of all node and edge stalks to d: each restriction map will have dimensions $d \times d$, and the sheaf Laplacian matrix will have dimensions $nd \times nd$. The sheaf Laplacian can be seen as a generalization of the well-known graph Laplacian on G: if we define a trivial sheaf where each stalk is isomorphic to $\mathbb{R}$ and the restriction maps are the identity, we recover the simple $n \times n$ graph Laplacian from the sheaf Laplacian.

2.3 Neural Sheaf Diffusion

Suppose to consider a graph $G = (V, E)$ and let $\mathbf{x} \in C^0(G; F)$ be an nd-dimensional vector obtained by stacking vertically n d-dimensional feature vectors $\mathbf{x}_v \in F(v)$, one for each node $v \in V$. If for each feature vector we allow f different channels, we obtain the feature matrix $\mathbf{X} \in \mathbb{R}^{nd \times f}$, of which the columns are vectors in $C^0(G; F)$.

Sheaf diffusion is a process on $(G; F)$ governed by the following differential equation:

$$\mathbf{X}(0) = \mathbf{X}, \dot{\mathbf{X}}(t) = -\Delta_F \mathbf{X}(t) \tag{3}$$

that can be discretized via the explicit Euler scheme with unit step-size:

$$\mathbf{X}(t+1) = \mathbf{X}(t) - \Delta_F \mathbf{X}(t). \tag{4}$$

In particular, the discrete diffusion model used by Bodnar et al. in (1) adapts (4) in the following way:

$$\mathbf{X}_{t+1} = \mathbf{X}_t - \sigma(\Delta_{F(t)}(\mathbf{I}_n \otimes \mathbf{W}^t_1)\mathbf{X}_t\mathbf{W}^t_2). \tag{5}$$

where $\mathbf{W}^t_1$ and $\mathbf{W}^t_2$ are weight matrices, and the restriction maps that define $\Delta_{F(t)}$ have a pre-defined and learnable parametric form $F_{v \trianglelefteq e:=(v,u)} = \Phi(\mathbf{x}_v, \mathbf{x}_u)$, on which constraints such as being diagonal or orthogonal can be imposed. It's important to highlight that both the sheaf $F(t)$ and the weight matrices are time dependent: different parameter values for $\mathbf{W}^t_1$, $\mathbf{W}^t_2$ and $\Delta_{F(t)}$ are learnt at each layer.

3 NSD for Graph Classification

One question addressed in this project is whether the benefits brought by sheaves in the context of node classification can be useful also for the task of graph classification. To come up with an answer, the *Neural Sheaf Diffusion* (1) procedure was adapted in different ways.

Once meaningful node features are computed, the simplest idea would be to equip the model with standard permutation-invariant readout operations, such as mean, sum, max and their concatenation. These methods are exactly what I implemented in the first place, and it is interesting to observe how their performance can substantially differ according to the task and dataset at hand.

Additionally, driven by the idea that complex geometric structures like sheaves may require more sophisticated readout operations, the experiments were taken further. Considering the analysis carried out by Grattarola et al. (5) on pooling methods for GNNs, I wanted to supply the model with an already existing pooling technique that may suit the problem. To this end, I opted for Self-Attention Graph Pooling (6), because sparse pooling methods had demonstrated to generally achieve a better performance with respect to the dense ones on the datasets I had chosen for my experiments, namely Enzymes and Mutagenicity. In particular, I adapted and implemented both the global and the hierarchical pooling architectures described in the SAGPool paper.

The last readout method I tested on the model was inspired by the recent paper by Buterez et al. (7) on adaptive readouts for GNNs. According to their findings, it may be the case that in some particular settings the ability of readouts to fit perfectly the task at hand may be more important than their invariance to permutations. When this happens, non-permutation-invariant modules like MLPs can strongly outperform standard readout operations such as mean or sum.

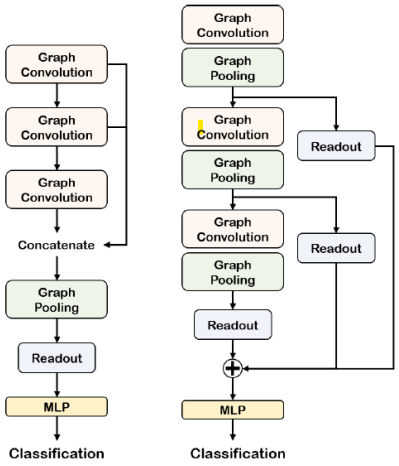

Figure 1: Global (left) and hierarchical (right) SAG pooling models. This scheme is taken from the original Self-Attention Graph Pooling paper.

Therefore, I decided to also experiment this approach by using the MLP readout layer proposed by Buterez et al. (7), also motivated by the astonishing results they obtained with it specifically on the Enzymes dataset.

4 NSD and Edge Features

Bodnar et al. in (1) decided to implement sheaf diffusion by pre-defining the $d \times d$ restriction maps with a parametric and learnable form: $F_{v \trianglelefteq e:=(v,u)} = \Phi(\mathbf{x}_v, \mathbf{x}_u)$, where $\Phi : \mathbb{R}^{d \times 2} \to \mathbb{R}^{d \times d}$ and, in practice, $\Phi(\mathbf{x}_v, \mathbf{x}_u) = \sigma(W[\mathbf{x}_v || \mathbf{x}_u])$. Something that this definition doesn't take into account are the potential edge features that may be associated to e, and that in principle should somehow influence the restriction maps among $F(v)$, $F(u)$ and $F(e)$. Additionally, edge features may be important to consider for certain graph classification tasks. For this reason I came up with some ways to integrate their information into the restriction maps, through a process that is activated only when non-trivial edge features are present in the considered dataset.

The first step simply consists in mapping, through a non-linear transformation, the original edge features into d-dimensional vectors $\mathbf{x}_e$, ideally belonging to $F(e)$. The stalk dimension is the same as for the vertexes, while the number of feature channels in this case is set to 1 because it has proven to work better in practice. This implies that the features from the node and edge stalks have different dimensions: $\mathbf{x}_e \in \mathbb{R}^{d \times 1}$ while $\mathbf{x}_v, \mathbf{x}_u \in \mathbb{R}^{d \times f}$. Even though for simplicity the equations defining $F_{v \trianglelefteq e:=(v,u)}$ use a single feature channel, in practice all channels are supplied as input. The difference in number of feature channels is quite relevant because it also influences the operators' dimensions, but it won't be emphasized further for ease of notation.

I tried handling the edge information in three different ways:

- **Concatenation.** This simply consists in extending the concatenation of node features also to edge features: $F_{v \trianglelefteq e:=(v,u)} = \Phi(\mathbf{x}_v, \mathbf{x}_u, \mathbf{x}_e) = \sigma(W[\mathbf{x}_v || \mathbf{x}_u || \mathbf{x}_e])$
- **Linear transformation.** Setting $\mathbf{z} = \Phi(\mathbf{x}_v, \mathbf{x}_u) = \sigma(W[\mathbf{x}_v || \mathbf{x}_u])$, the parametric restriction map is redefined as: $F_{v \trianglelefteq e:=(v,u)} = \mathbf{z} + \Psi(\mathbf{z}, \mathbf{x}_e)$, where $\Psi(\mathbf{z}, \mathbf{x}_e) = \sigma(P[\mathbf{z} || \mathbf{x}_e])$. P is an additional learnable matrix.
- **Bilinear transformation.** This case is similar to the previous one, with the difference of performing a bilinear product instead of a linear one. We set $\mathbf{z} = \Phi(\mathbf{x}_v, \mathbf{x}_u) = \sigma(W[\mathbf{x}_v || \mathbf{x}_u])$. The parametric restriction map is now redefined as: $F_{v \trianglelefteq e:=(v,u)} = \mathbf{z} + \Psi(\mathbf{z}, \mathbf{x}_e)$, where $\Psi(\mathbf{z}, \mathbf{x}_e) = \sigma(\mathbf{z} Q \mathbf{x}_e)$. Again, Q is an additional learnable matrix.

5 Experiments

In order to validate the previously exposed ideas, multiple experiments were carried out. The first step was choosing the datasets on which to perform the comparisons in a way that would be informative enough. The second choice regarded the architectures, that is the combination of the components proposed in Section 3 and 4, on which to execute the tests. Eventually, a pair of state-of-the-art models for graph classification was selected and tested on the same datasets to set some benchmarks.

For all experiments, the number of epochs was set to a maximum of 200, and early stopping with patience of 15 steps was adopted, by monitoring the performance on the validation set. The learning rate and the readout probabilities were adjusted to the specific models in order to grant the best possible results. The number of layers was set to 4 for all models, except for hierarchical SAGPool, in which it was reduced to 2 in order to better deal with the presence of the additional pooling layers between the diffusion ones.

5.1 Datasets

For the tests Enzymes and Mutagenicity were used, taken from the TUDataset collection (8). Since they have quite different properties, they allow for variability during experiments, and in both cases the limited average number of nodes lightens computations. Enzymes is a dataset from the domain of Bioinformatics and it is composed by 600 graphs with 3-dimensional node labels and no edge labels. The goal is classifying enzymes in their correct category, choosing among six. For what concerns Mutagenicity instead, it's a chemical compound dataset of drugs, that can be categorized into two possible classes: mutagen and non-mutagen. In this case, the number of graphs is 4337 and they are equipped with 14-dimensional node features and 3-dimensional edge features.

	Enzymes	**Mutagenicity**
Domain	Bioinformatics	Small molecules
N. of graphs	600	4337
Classes	6	2
Avg nodes	32.63	30.32
Avg edges	62.14	30.77
Node features	3	14
Edge features	0	3
Train/val/test split	440/80/80	3700/300/337

Table 1: Summary of the statistics related to datasets

5.2 NSD Models

Different combinations of readout layers and edge-feature-handling techniques were used for experiments. Since the total number of possible combinations would have been too high, a pre-defined procedure was followed to select the models to test, in order to also discard in advance the ones that most likely wouldn't lead to promising results and reduce the computational overhead.

For both datasets, the investigations followed this scheme:

1. The first goal was to determine the most suitable type of restriction map. In this phase the readout operation was kept fixed, namely the concatenation of mean, sum, and max. The tested models were:
   - Diagonal sheaf diffusion model, with $d = 1$. This means, in practice, considering a *scalar* GNN model, with sheaves having trivial stalks and identity restriction maps. This can be considered as a baseline.
   - Diagonal sheaf diffusion model, with $d = 3$. Differently from before, the sheaf's stalks and restriction maps are no more trivial in this case, hence it is possible to analyze how the higher dimensionality affects the performance.
   - Bundle sheaf diffusion model, with $d = 3$. The restriction maps are now required to satisfy an orthogonality constraint, instead of just having a diagonal structure. This allows for more expressiveness in the computation of node-level embeddings.
2. The second stage of the experiments was carried out for datasets with non-null edge features, that is Enzymes only. The model with non-trivial sheaf structure achieving the best result in the first phase was taken and equipped with the three edge-feature-handling techniques explained in Section 4: concatenation, linear transformation and bilinear transformation.
3. The last set of tests regarded the analysis of the readout and pooling techniques proposed in Section 3. Again, the best performing model from the second (or first, in case of trivial edge features) phase was considered, and on top of it the following readout techniques were tried out sequentially: mean, sum, max, MLP, global SAGPool and hierarchical SAGPool.

5.3 Benchmark Models

In order to effectively evaluate the behavior of the designed architectures, their results were compared with the ones obtained by two state-of-the-art models for graph classification: Hierarchical Graph Pooling with Structure Learning (HGP-SL) (9) and Norm-GN (10). The choice is justified by the top results achieved by those models on Enzymes and Mutagenicity, as reported on the Leaderboard section of PapersWithCode. It should be noticed that the results I obtained by re-executing the official codes mostly align with the ones claimed on the Leaderbord, except for HGP-SL on Enzymes: the final accuracy I achieved is significantly lower (51.7) with respect to the official one (68.8).

6 Results

The results in accuracy obtained in the study are reported in the following table. The models were tested following the scheme proposed in Section 5.2, and it shows that for both datasets the best architecture from the first phase was the diagonal one, hence such restriction map was adopted for the second and third stages. In the case of Mutagenicity, *bilinear e.h.* resulted being the best edge-handling technique, hence it was used also during the third phase.

	Enzymes	**Mutagenicity**
Scalar baseline (d = 1)	40.0	78.0
Diag	42.5	78.0
Bundle	40.0	76.9
Diag + concat e.h.	-	77.8
Diag + linear e.h.	-	76.9
Diag + bilinear e.h.	-	78.6
Diag + mean readout	45.0	75.4
Diag + sum readout	30.0	76.3
Diag + max readout	31.3	80.4
Diag + MLP readout	50.0	77.2
Diag + global SAG	45.0	75.1
Diag + hierarchical SAG	35.0	76.6
HGP-SL	51.7	81.8
Norm-GN	72.0	83.7

Table 2: Accuracies obtained on the two benchmark datasets

It seems that, in general, the introduction of sheaves for graph classification doesn't provide substantial benefits to standard GNNs. This is suggested by the low difference in accuracy between the case $d = 1$ and $d > 1$. Furthermore, equipping the SNN with different sorts of edge-feature-handling and readout techniques is still not enough to grant better results with respect to the ones of SOTA architectures. The fact that the diagonal restriction map outperforms the orthogonal one on both datasets is probably due to the higher complexity and number of parameters for the bundle case, and that this complexity cannot be properly exploited in terms of expressiveness as it happens in the task of node classification.

It is quite clear that the edge-feature-handling techniques do not bring strong improvements, and even when they do (in the bilinear case) they are still not too significant. One reason may be that they tackle the problem of edge feature integration in a too simplistic way, while more complex and theoretically grounded methods would be probably needed.

Interestingly, instead, considering different readout functions appears to have a stronger impact on how performance varies. In the case of Enzymes, indeed, the MLP distinctly outperforms all permutation-invariant methods. This behavior could be expected, to some extent, from the promising results already achieved by MLP readouts on Enzymes reported in (7). In this case the ability of readout to fit perfectly the task at hand results to be more important than preserving permutation invariance. This does not hold for Mutagenicity for which, instead, a standard max readout appears to be the best option. Finally, for SAGPool the choice between the global and hierarchical structure seems to be strictly de-

pendent on the dataset and not determined by unconditioned general features.

7 Conclusion

The primary goal of this work was to investigate the potentiality of sheaf diffusion in the context of graph classification. The study was carried out by exploring multiple pooling and readout operations, from standard ones to non-permutation-invariant ones, in order to find out the best approach to deal with such complex geometric structures. Additionally, an attempt was made, through different approaches, to integrate the edge feature information into the layer-wise computation of the restriction maps. The obtained results were not astonishing when compared to the performance of SOTA architectures. This may lead to the conclusion that sheaf structures don't bring to graph classification substantial benefits, or that maybe more advanced and ad-hoc pooling techniques, as well as edge-feature-handling methods, should be further studied and designed in order to better exploit them in the task.

8 Hints for future work

In this section I would like to list some hints and ideas that arose during the development of the project; they may be not so easy to investigate in the short run, but they could possibly be interesting directions for future research.

- As suggested by Barbero et al. (11) it would probably be much better if, instead of learning the sheaf's parameters from data through SGD, there was an efficient way to pre-compute the sheaf structure manually.
- I tried integrating the information given by edge features into the definition of the parametric restriction maps, but the efforts didn't bring satisfactory results. It could be interesting to think of more sophisticated techniques than the proposed ones, that may perform better in practice.
- To enhance the performance of sheaves in graph classification settings, one option could be considering higher order graph structures at different scales, devising k-dimensional Sheaf Neural Networks.
- A further field of investigation regards how the geometrically complex sheaf structure could support novel techniques for interpretability and explainability in GNNs.

References

[1] Bodnar, C., Di Giovanni, F., Chamberlain, B.P., Liò, P., and Bronstein, M. (2022). Neural Sheaf Diffusion: A Topological Perspective on Heterophily and Oversmoothing in GNNs. *arXiv:2202.04579*

[2] Hansen, J. and Gebhart, T. (2020). Sheaf Neural Networks. *arXiv:2012.06333*

[3] Curry, J. M. (2014). Sheaves, cosheaves and applications.

[4] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*

[5] Grattarola, D., Zambon, D., Bianchi, F.M. and Alippi, C. (2021). Understanding Pooling in Graph Neural Networks. *arXiv:2110.05292*

[6] Lee, J., Lee, I. and Kang, J. (2019). Self-Attention Graph Pooling. *arXiv:1904.08082*

[7] Buterez, D., Janet, J.P., Kiddle, S.J., Oglic, D. and Liò, P. (2022). Graph Neural Networks with Adaptive Readouts. *arXiv:2211.04952*

[8] Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. (2020). Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv:2007.08663*

[9] Zhang, Z., Bu, J., Ester, M., Zhang, J., Yao, C., Yu, Z. and Wang, C. (2019). Hierarchical Graph Pooling with Structure Learning. *arXiv:1911.05954*

[10] Yang, M., Shen, Y., Li, R., Qi, H., Zhang, Q. and Yin, B. (2022). A New Perspective on the Effects of Spectrum in Graph Neural Networks. *arXiv:2112.07160*

[11] Barbero, F., Bodnar, C., de Ocáriz Borde, H.S., Bronstein, M. Veličković, P. and Liò, P. (2022). Sheaf Neural Networks with Connection Laplacians *arXiv:2206.08702*