[Re] PolyGCL: Graph Contrastive Learning Via Learnable Spectral Polynomial Filters

GDL 2024
Group id: 2
Project id: PolyGCL

Michele Cattaneo, Nicolai Hermann, Oliver Tryding
{michele.cattaneo, nicolai.hermann, oliver.tryding}@usi.ch

Abstract
Graph contrastive learning has gained significant attention for learning effective representations from unlabeled graph data. In this paper reproduction, we replicate and evaluate the methodology proposed by the PolyGCL pipeline, which uses spectral polynomial graph filters to achieve contrastive learning by learning a linear combination of a high-pass and low-pass view. The original paper studies the necessity of introducing high-pass information when a graph presents a high degree of heterophily, that is, when nodes connected by an edge are likely to have different labels, which poses a challenge for current unsupervised learning approaches. Through this replication study, we assess the reproducibility and robustness of the method using both their open-source code and our own implementation, discuss implementation details, and provide insights into the practical considerations and challenges of applying self-supervised contrastive learning.

1 Introduction

This project addresses the problem of self-supervised graph representation learning when dealing with both heterophilic and homophilic graphs. Heterophily is the tendency in graph data to have edges between nodes that do not share the same labels. The inverse tendency is called homophily. Existing methods rely on low-pass graph convolutional filters which underperform when the homophily assumption is not met, due to the smoothing effect on node features. Graph contrastive learning has become a predominant approach to self-supervised learning on graphs, which entails the training of an encoder to be contrastive with respect to data that have statistical dependencies and those that do not, by producing representations that are easily distinguishable when comparing positive and negative examples. With this work we assess the reproducibility of PolyGCL [1], in which this problem is approached from a spectral point of view by learning both a high-pass and low-pass view and coefficients weighing the two contributions to produce a final embedding. To do that, a contrastive objective function is devised. Initially, the code was not open-sourced and we implemented a first model based solely on the explanations contained in the paper. In a second phase, after the code was published, we compared our code with the original implementation and adapted it in order to produce fair comparisons. For the most part, we were able to reproduce the claimed results on a series of real-world and synthetic datasets, spanning a wide range of heterophily and homophily degrees. We start by providing a brief but necessary background on the topic, followed by the related work and methodology details of PolyGCL. We then discuss our implementation, the difficulties we encountered, and the differences between the implementation and the method described in the paper. Finally, we report and discuss the obtained results.

2 Preliminary

We will consider attributed graphs $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ of size N where $\mathcal{X} = \{(i, x_i)\}_{i=1:N}$ is the set of nodes and their attribute vector, and $\mathcal{E} = \{(i, j)\}$ is the set of edges, which we assume not to have any attributes. Although nodes naturally do not take any order, it is useful to consider $X \in \mathbb{R}^{N \times d}$ to be the matrix of ordered node attributes. We will then define $Z = F(X, A)$ the output of a graph neural network F, given the node attribute matrix and the adjacency matrix A, which is an ordered representation of $\mathcal{E}$, such that $A_{ij} = 1 \leftrightarrow (i, j) \in \mathcal{E}$.

Graph neural networks (GNNs) are permutation equivariant functions, i.e. functions F such that $F(PX, PAP^T) = PF(X, A)$ for any permutation matrix P, that computes a meaningful latent representation of node attributes. To do that they internally use permutation invariant functions $z_i = \phi(x_i, X_{\mathcal{N}(i)})$ to compute the representation of node i given the neighbouring features $X_{\mathcal{N}(i)}$ [2]. GNNs can be divided into *spatial* ([3] [4] [5]) and *spectral* ([6] [7] [8]) GNNs, depending on whether they perform convolutions in the spatial or spectral domain. This project focuses on spectral GNNs, which are based on the spectrum of the Laplacian matrix. The Laplacian of a graph is defined as $L = D - A$, where $D = \text{diag}(A\mathbf{1})$ is a diagonal matrix with the degrees of the nodes. The normalized Laplacian is defined as $\tilde{L} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ and is diagonizable as $\tilde{L} = U \Lambda U^T$, where U is the Fourier basis of the graph and $\Lambda = \text{diag}([\lambda_0, \dots, \lambda_{N-1}])$. A graph signal $x \in \mathbb{R}^N$ of graph $\mathcal{G}$ can be transformed in the graph Fourier space by $\hat{x} = U^T x$, while the inverse transform is defined as $x = U\hat{x}$ [9]. A graph can then be filtered by a filter g_θ as $y = U g(\Lambda) U^T x$ for a filter $g_\theta(\Lambda)$. ChebNet from [6] approximates g_θ by a truncated expansion $g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$, where $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I$, T_k is the Chebyshev polynomial of order k and θ_k are the Chebyshev coefficients. The filtering operation is then defined as:

$$y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x \quad (1)$$

where $T_k(\tilde{L})$ is the Chebyshev polynomial of k-th order evaluated on the scaled Laplacian $\tilde{L} = 2L/\lambda_{max} - I$. The Chebyshev polynomials can be recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$. Then the overall structure of ChebNet is:

$$\begin{aligned} Z &= \sum_{k=0}^{K} P^{(k)}\Theta^{(k)} \\ P^{(1)} &= X \\ P^{(2)} &= \tilde{L} \\ P^{(k)} &= 2\tilde{L}P^{(k-1)} - P^{(k-2)} \end{aligned} \quad (2)$$

with learnable parameters $\Theta^{(k)}$.

Contrastive methods are a common approach for unsupervised learning, where representations are learned by training an encoder to be *contrastive*. This involves distinguishing between representations that are dependent on each other and those that are not. In practice, a contrastive approach may involve training an encoder to increase the similarity score for pairs of real inputs and decrease the score for pairs involving fake inputs. By doing so, the encoder learns to produce similar embeddings for related inputs while ensuring that unrelated inputs are mapped to distant regions in the embedding space [10]. This framework is particularly useful in graph contrastive learning, where the objective is to learn robust node or graph-level representations by contrasting nodes or subgraphs that share structural or attribute similarities with those that do not.

3 Related works

Deep Graph Infomax (DGI) [10] introduces an objective for unsupervised graph representation learning based on *mutual information*, where mutual information contained in a global representation and local representations of the graph has to be maximized. The objective is to learn an encoder $\mathcal{E}$ producing node embeddings $Z = \mathcal{E}(X, A)$ such that they capture global information of the entire graph. To do that, a discriminator $\mathcal{D}$ should be able to differentiate between local representations z_i and corrupted representations (not belonging to the graph) $\tilde{z}_i$ when compared to a global summary of the graph $s = \mathcal{R}(Z)$. Because the encoder is a graph convolutional network that aggregates features of neighbouring nodes (a patch of the graph) into the features of the receiving nodes, each node contains part of the global information. This idea is also used in PolyGCL.

ChebNetII introduced in [11] was dealing with the unexpected empirical results showing ChebNet [6] performing worse than GCN [5], which is a simpler version of ChebNet, where only the first two Chebyshev polynomials are used. ChebNet's theoretical higher expressiveness is not reached in practice because illegal coefficients are learned by ChebNet approximating analytic filter functions, resulting in degraded performance. The authors propose a novel filtering based on Chebyshev interpolation, improving the original Chebyshev polynomial approximation, which is also used in PolyGCL.

4 Methodology

The graph convolutional layer in the PolyGCL method is an extension of the ChebNet filtering presented in Formula 1 in which polynomial filters are learned on two separate high-pass and low-pass channels, generating two distinct spectral views. Following [11], the graph convolution is approximated by Chebyshev polynomials with interpolation as base polynomials. Given a continuous filter function $g(\hat{\lambda})$, let $x_j = \cos\left(\frac{j+1/2}{K+1}\pi\right), j = 0, \ldots, K$ denote the Chebyshev nodes for T_{K+1}. The filter value $g(x_j)$ at the Chebyshev node x_j is re-parameterised as a learnable parameter γ_j and the overall filtering operation can be defined as:

$$Z = \sum_{k=0}^{K} \underbrace{\frac{2}{K+1} \sum_{j=0}^{K} \gamma_j T_k(x_j)}_{w_k} T_k(\tilde{L})X \quad (3)$$

Note that in this formulation, feature propagation and transformation are decoupled and features X can be first transformed by an MLP as $\tilde{X} = f_\theta(X)$ [11].

To achieve a low pass and a high pass filtering, a prefix sum and a prefix difference are applied on γ_j respectively. The result is that in the former case, the learnable parameters γ_j increment as j increases and in the latter case the value decreases. Let γ_0 be some initial value. Then let γ_i^H be the resulting prefix sum up to $j = i$ and γ_i^L be the result of the prefix difference up to $j = i$. That is:

$$\gamma_i^H = \sum_{j=0}^{i} \gamma_j, \quad \gamma_i^L = \gamma_0 - \sum_{j=1}^{i} \gamma_j, \quad i = 1, \ldots, K \quad (4)$$

As a consequence, $\gamma_i^H \leq \gamma_{i+1}^H$ and $\gamma_i^L \geq \gamma_{i+1}^L$, guaranteeing the high pass and low pass property of $g(\hat{\lambda})$ we want to approximate.

The encoders, providing the high pass and low pass views can therefore be defined as follows:

$$\begin{aligned} Z_L &= \mathcal{E}^L(X, A) = f_\theta\left(\sum_{k=0}^{K} w_k^L T_k(\tilde{L})X\right) \\ Z_H &= \mathcal{E}^H(X, A) = f_\theta\left(\sum_{k=0}^{K} w_k^H T_k(\tilde{L})X\right) \end{aligned} \quad (5)$$

where w_k^H is obtained by substituting γ_j with γ_j^H in $\frac{2}{K+1}\sum_{j=0}^{K} \gamma_j T_k(x_j)$. Similarly, w_k^L is obtained by substituting with γ_j^L. Function f_θ is an MLP with shared weights between both views.

In order to train this architecture in an unsupervised manner, we need an unsupervised objective function to optimize. The two views are first combined linearly as $Z = \alpha \cdot Z_L + \beta \cdot Z_H$, with α and β being learnable parameters. Following [10], contrastive learning is achieved by maximizing mutual information between "local patches" and global summaries of the graph. Local patches are node embeddings that depend on their neighborhood. Global summaries are achieved with a global graph pooling operation [12], which in this case is a simple mean pooling $g = \text{mean}(Z) = \frac{1}{N}\sum_{i=0}^{N} z_i$ for a graph of size

N. To obtain negative examples, the input features X are randomly shuffled, while the adjacency matrix is left untouched, obtaining $\tilde{X}$, and fed into the high pass and low pass encoders, obtaining $\tilde{Z}_H$ and $\tilde{Z}_L$. To score the mutual information between node and graph representations, a discriminator $\mathcal{D}$ is defined as $\mathcal{D}(z_i, g) = \sigma\left(z_i W g^T\right) \in (0, 1)$ with learnable parameters $W \in \mathbb{R}^{d \times d}$ for embeddings of size d. Finally, we can define the objective function as a binary cross-entropy function to minimize:

$$\mathcal{L}_{BCE} = -\frac{1}{4N}\sum_{i=1}^{N} \log \mathcal{D}(z_{i,L}, g) + \log(1 - \mathcal{D}(\tilde{z}_{i,L}, g)) + \log \mathcal{D}(z_{i,H}, g) + \log(1 - \mathcal{D}(\tilde{z}_{i,H}, g)) \tag{6}$$

Given the above discussion and definitions, training PolyGCL can be carried out by following the steps described in Algorithm 1, given a maximum number of iterations and having chosen the truncation length K.

Algorithm 1: PolyGCL Training

```
Data: Node features $X$, adjacency $A$
Result: Node encoders $\mathcal{E}^L$ and $\mathcal{E}^H$
Initialize $\mathcal{D}, f_\theta, \alpha, \beta, \gamma_i$ for $i = 0 \dots K$
for $epoch = 1 \dots T$ do
    Compute $\gamma_i^H$ and $\gamma_i^L$
    Compute $w_k^H$ and $w_k^L$
    $\tilde{X} \leftarrow \text{shuffle}(X)$
    $Z_L \leftarrow \mathcal{E}^L(X, A), Z_H \leftarrow \mathcal{E}^H(X, A)$
    $\tilde{Z}_L \leftarrow \mathcal{E}^L(\tilde{X}, A), \tilde{Z}_H \leftarrow \mathcal{E}^H(\tilde{X}, A)$
    $Z \leftarrow \alpha Z_L + \beta Z_H$
    $g \leftarrow \text{mean}(Z)$
    loss $\leftarrow \mathcal{L}_{\text{BCE}}(g, Z_L, Z_H, \tilde{Z}_L, \tilde{Z}_H)$
    Optimize $\mathcal{D}, f_\theta, \alpha, \beta, \gamma_i$
end
```

5 Implementation

In this section, we will describe our implementation of the paper's methods and compare it to the authors' code that was later made open source. We will further outline the differences and inconsistencies between the two implementations and the methods described in the paper.

5.1 Reproduction

The reproduction consisted of multiple modules:

- The high and low pass encoder $\mathcal{E}$.
- The loss computation detailed in Eq. 6.
- Evaluation by running logistic regression on the generated embeddings.
- The overall training algorithm, given in Algorithm 1 of their paper.
- The synthetic generation of the cSBM datasets.

Encoder: Our encoder was based on Pytorch Geometric's implementation of ChebNet*, which we adapted to implement ChebNetII's Chebyshev interpolation, which in turn provides the base for PolyGCL's graph convolution. Following the paper, we utilized a single set of trainable parameters $\gamma_{1:K}$ and initial value γ_0 shared between the high pass and low pass view of the encoder. To guarantee that the values of the various γ_i remain positive, we apply a ReLU activation on their learned values. In order to freely choose the dimension onto which we apply the convolutions, we initially encode the input features with a linear layer followed by a dropout layer. We then apply the graph convolution on the embedded features, apply another dropout, followed by a batch normalization and a linear layer with non linear activation. In a subsequent subsection, we will explore the choice of specific hyperparameters, such as dropout rates, learning rates, embedding dimension, and activation functions.

Loss: The loss function was initially implemented by explicitly computing Equation 6, which turned out to be problematic. The discriminator, implemented as a bilinear layer with sigmoid activation, when trained could produce 1.0 or 0.0 as outputs, as the numerical value of sigmoid can quickly reach its upper and lower limit sufficiently close, and evaluating $\log(0)$ or $\log(1 - 1.0)$ would produce invalid loss values. We considered adding a small value ϵ to any logarithm to avoid the problem but ended up relying on Pytorch's built-in cross-entropy loss. For that, we provided labels set to 1 for Z_L and Z_H, and labels set to 0 for $\tilde{Z}_L$ and $\tilde{Z}_H$.

Evaluation: Evaluating the quality of node representations alone is not trivial. Following the experiments carried out in the paper, we evaluate the quality of the representation by assuming that if they are good, a simple linear classifier should be able to have a high accuracy on them. For this, we wrote a small Pytorch training loop for a logistic regression model. Following [1], we use random train/test/validation splits in ratios of 60/20/20. However, we make an exception for the datasets *roman_empire*, *amazon_ratings*, *minesweeper*, *tolokers*, and *questions* as they were imbalanced in terms of labels distribution. This led to biased classifiers that achieved high test accuracy before even training the embedding. Using the provided splits from PyTorch geometric solved the problem for *roman_empire* and *amazon_ratings*. Following the authors, we replaced accuracy with the more appropriate area under the receiver operating characteristic curve (AUC-ROC) for imbalanced classification for *minesweeper*, *tolokers*, and *questions*. To optimize we used `Adam` and `CrossEntropy` or `BinaryCrossEntropy` for multiclass and two class datasets respectively. Early stopping was applied to prevent overfitting.

At the end of the encoder's training process we load the best encoder model achieving the lowest unsupervised loss again. We evaluate it with the post-evaluation script provided by the authors. The code fits a logistic regression model to 10 different train/test/validation splits to get an uncertainty estimate. For a fair comparison, we compare our implementations to the results of their evaluation script.

Training Algorithm: To reproduce all experiments we wrote a dataset factory to load the various datasets from `torch_geometric.datasets` and `ogb`. The training

*`https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/nn/conv/cheb_conv.html#ChebConv`

loop is a classical Pytorch training loop where we logged α, β, $\gamma_{1:K}$, and the logistic regression results to Weights and Biases to track our experiments. We used `Adam` as our optimizer.

Synthetic Dataset: To control the degree of homophily/heterophily of a dataset the authors proposed a way to generate synthetic datasets. The workings are detailed in their Appendix C.1 which we reproduced. However, since their code got published before we started running the experiments we ended up using their implementation for better comparability.

5.2 Authors' code

The authors implemented their method with Pytorch 1.11.0 and torch-geometric 1.7.2. They divided their codebase into subfolders containing independent implementations of their method for each experiment. Therefore a lot of code is repeated. When comparing each implementation we found changes in the model architecture for different datasets. For example, the use of the Laplacian was different as sometimes the GCN norm $(D+I)^{-1/2}(A+I)(D+I)^{-1/2}$ was used instead of the ChebNet norm $2(I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}})/\lambda_{max} - I$, for which we confronted them and they argued the impact on the spectral domain is minimal to a certain extent. Also the datasets *chameleon*, *citeseer*, *cora*, *cornell*, *actor*, *pubmed*, *squirrel*, *texas*, and *wisconsin* were included alongside the code as well. For the heterophilic datasets *chameleon* and *squirrel* we were unable to load the data they provided because of an incompatibility issue with the PyG version that we could not resolve after many reinstallation attempts on different machines. Interestingly, after rewriting the data loader using the datasets directly provided from a newer PyG version, the observed performance of the authors' code strongly contradicted the performance claimed in the paper. The authors claim that they used the same dataset version as [13], [14], and [15]. Since *chameleon* and *squirrel* were shown to have data leakage in their test set[16], the authors argued that the performance degradation may be attributed to that, as newer versions of PyG (2.5.2) have not resolved this issue. However, they referred us to using the data that they provided which we couldn't load nor verify. They further pointed out that for their experiments, the graphs of these two datasets were made undirected. We tried to do the same, however the performance did not improve. Either way, to further test their method on heterophilic datasets, the authors of [16] proposed five new heterophilious benchmarks. The results for those datasets are depicted in Table 6.

5.3 Differences

After the authors published the final version of the paper along with their code base we started comparing our implementations which unveiled significant differences upon which we contacted the authors to understand their choices better. In the paper they claimed that $\gamma_0^H = \gamma_0^L = \gamma_0$ but used different initial values for γ_0^H and γ_0^L. Sometimes they also set them as trainable parameters, for other datasets they were fixed. Initially, we had them fixed which caused detrimental dips in accuracy. Fixing them to different values almost doubled the measured performance. Furthermore, they used two different sets of parameters γ for the high and low pass respectively instead of a shared set of γs, as the paper seems to suggest. They argued that that changing the initial values or having separate sets of γ does not affect the theoretical aspect of their work since the high-pass and low-pass properties are still ensured by the prefix sum and prefix difference, and that all these scenarios boil down to a special case of the algorithm described on the paper, which we agree on. As long as each γ_i is non-negative the prefix sum ensures a monotonic increase $\gamma_i^H \leq \gamma_{i+1}^H$ and the prefix difference ensures the monotonic decrease $\gamma_i^L \geq \gamma_{i+1}^L$. Therefore the filter properties are still satisfied. Also having two separate sets of γ will only give the model more expressive power.

However, setting γ_0 to some non-zero constant forces the model to use high-frequency information as $\gamma_{1:K}^H$ can only increase. Similarly, for $\gamma_{1:K}^L$ the model would need to learn to very quickly decrease γ in order to ignore low-frequency information. For example, forcing high-pass filters on very homophilous datasets can hinder training as it adds noise to the embedding. Likewise for highly heterophilious datasets, biasing the model also to include low-frequency signals can dilute the training progress. If the model wants to ignore high- or low-frequency information it would need to adapt α and β respectively which might be a difficult interaction to learn.

Furthermore, the authors used different learning rates for the linear layers compared to the convolution, α, and β. Typically the linear layers had much smaller learning rates compared to the other components. We adapted this training scheme and optimized both learning rates.

In some of their convolution implementations, they applied ReLU after computing the prefix difference to enforce the positivity of $\gamma_{0:K}^L$, in others negative values were possible. Adding a ReLU after the prefix difference was not described in their paper even though it makes a lot of sense. We would expect that if γ_i^H becomes very negative, the low-pass filter could start including high-frequencies again. The authors agreed with us and BernNet[17] also claimed that negative filter values are considered well-defined. Therefore it becomes difficult to determine whether a filter function with both positive and negative values is a low-pass or high-pass filter. But interestingly when observing γ^L on our runs, the values never seemed to become negative. Just to be sure we added the ReLU after commuting the prefix difference.

5.4 Datasets

For our experiments, all real-world datasets were obtained by downloading them from Pytorch Geometric's datasets module and their features were normalized. In case the data provided by the authors was loadable we used it to reproduce their results. A short description of the datasets can be seen in Table 1.

5.5 Hyperparameters

Since we have two implementations, our own and the later published code from the authors, we evaluated two-fold. Firstly, we reran the authors' code with the run configuration they provided in their codebase. Those hyperparameters were obtained using grid search according to their Appendix F, table 9-11. Results obtained with the same hyperparameters with our implementation were not on par with their model.

Table 1. Datasets

	Number of nodes	Number of edges	Number of classes
CORA [18]	$2,708$	$5,429$	7
CITESEER [18]	$3,327$	$4,732$	6
PUBMED [18]	$19,717$	$44,338$	3
CORNELL [19]	195	298	5
TEXAS [19]	187	325	5
WISCONSIN [19]	265	515	5
ACTOR [19]	$7,600$	$33,316$	5
CHAMELEON [20]	$2,700$	$36,101$	5
SQUIRREL [20]	$5,201$	$217,073$	5
ROMAN-EMPIRE [21]	$22,662$	$32,972$	18
AMAZON-RATINGS [21]	$24,492$	$93,050$	5
MINESWEEPER [21]	$10,000$	$39,402$	2
TOLOKERS [21]	$11,758$	$519,000$	2
QUESTIONS [21]	$48,921$	$153,540$	2

Since we have an additional embedding layer and actually train the batch norm, which was, probably due to an oversight, left not trainable in the original code, we decided to run hyperparameter sweeps for our model for fair comparison. The hyperparameter sweep configurations can be found under `sweep_configs` in our repository. All sweeps together ran about 96 hours.

5.6 Experimental setup

The experiments were run on machines equipped with an Nvidia RTX 4070, an Nvidia RTX 3080 and an Nvidia RTX 3090, with 12, 10, and 24 GB of memory respectively. The code has been developed in Python using the open-source libraries PyTorch [22] and PyTorch Geometric [23]. The code is available on GitHub.

5.7 Computational requirements

The computational requirements are summarized in Table 2 and Table 3. It is important to note that some hyperparameters that affect training time and memory usage used across the experiments are not consistent. The reported values are collected from the best-performing models and we refer to the hyperparameter choice for the specific runs to put these values into perspective. The chosen hyperparameters can be found in the GitHub repository.

6 Results

In the following section, we report empirical results on a wide variety of datasets. To better visualise the quality of the results (presented as accuracy or area under the receiver operating characteristic curve), we devised a simple four-coloring scheme, depicting in dark green results that are better or have an error within 2, in light green results with an error between 2 and 5, in light red errors between 5 and 10 and in dark red error higher than 10, of the respective metric.

Error	≥ 10	5 to 10	2 to 5	≤ 2

Table 4. Coloring scheme to present the results according to the error with respect to the claimed results.

6.1 Synthetic Dataset

In this section, we performed experiments on synthetic datasets generated with the cSBM model [24], following [15]. This model allows for the creation of graphs with a set degree of homophily, determined by a parameter $\phi \in [-1, 1]$, where a value of 1 indicates the highest degree of homophily and -1 the highest degree of heterophily. Table 7 shows the accuracy claimed in the authors' paper, compared to the results obtained by running the authors' code with their chosen hyperparameters and the results obtained by running our code with our hyperparameters. We can see that apart from the graph generated with $\phi = -1$, we obtained the expected results within a reasonable margin for the authors' code.

In Figure 1 we report the learned values of α, the value that controls the importance of the low pass encoder's view, as the level of homophily changes for the synthetic datasets. For this experiment, α was kept in the $[0, 1]$ range, and β was set to $1 - \alpha$, to ensure $\alpha + \beta = 1$. We can see a clear correlation between the value of α and the level of homophily, since as ϕ increases, so does α. As ϕ approaches 0, the structural information becomes useless. This is reflected in the values of α and β staying around 0.5, representing equal importance of the low and high pass filter view. In this setting, the performance is poor, which is expected and also reflected in supervised settings [15].

In Figure 2 we report the learned values of γ_i^H, γ_i^L, w_k^H and w_k^L for the PolyCGL convolution, using $K = 10$ and $\alpha + \beta = 1, \alpha \geq 0, \beta \geq 0$. On the horizontal axis, we have the training step and on the vertical axes a histogram showing the distribution of the K values. When $\phi = 1$, indicates a high homophility, we can clearly note that the high pass filter is understood to not be useful by the learning procedure since all weights are pushed towards 0. In the opposite setting, when the graph presents a high degree of heterophility with ϕ set to -1, we can see that while a low pass contribution is still present, the contribution of the high pass view in clear, depicted by the wide range of learned γ_i^H that consecutively define w_k^H. While the low pass contribution is still present, from Figure 1 we however know that for $\phi = -1$ the the low pass contribution is dampened by the learned value of α being low and close to 0, which in turns means that the model learned to use high pass filters when the graph is heterophilic.

In Table 7 we also report the results we obtained with our own implementation of PolyGCL. For $\phi = -0.25$, $\phi = 0$, and $\phi = 0.25$, we note the highest difference in performance, but we still are within 5 accuracy percentage points away from the claimed ones. The reproduction with $\phi = -1$ was problematic with the original code, however, with our own implementation, we obtained a very similar result to the claimed one. All other results are in line with the expected one. We can therefore claim that, overall, the reproducibility on synthetic data was successful and PolyGCL proved to be working well.

Table 2. Computational requirements for the various experiments. Superscripts [1] and [2] represent the statistics for the RTX 4070 and RTX 3090 respectively.

	CORA[2]	CITESEER[2]	PUBMED[2]	CORNELL[2]	TEXAS[2]	WISCONSIN[2]	ACTOR[2]	CHAMELEON[1]	SQUIRREL[1]
Memory (GB)	$\sim$ 24	$\sim$ 18	$\sim$ 24	$\sim$ 22	$\sim$ 11	$\sim$ 20	$\sim$ 24	$\sim$ 12	$\sim$ 12
GPU Time	31m03s	2m28s	140m36s	1m43s	50s	1m14s	98m49s	1m35s	20m29s

Table 3. Computational requirements for the various experiments on the heterophilic graphs and synthetic graphs. Superscripts [1], [2], and [3] represent the statistic for the RTX 4070, RTX 3090 and RTX 3080 respectively. Since the cSBM datasets are so similar they will be reported in a single column.

Methods	ROMAN-EMPIRE[2]	AMAZON-RATINGS[2]	MINESWEEPER[1]	TOLOKERS[2]	QUESTIONS[2]	cSBM[3]
Memory (GB)	$\sim$ 24	$\sim$ 10	$\sim$ 12	$\sim$ 19	$\sim$ 18	$\sim$ 10
GPU Time	9m17s	4m34s	1m50s	14m54s	5m38s	$\sim$ 25m

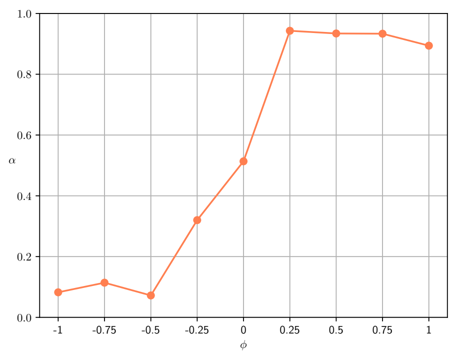


Figure 1. Values of α, which control the importance of the low pass encoder's view, as the level of homophily changes for graph coming from the cSBM model.

6.2 Real-world Dataset

In Table 5 we report the results of experiments on real-world datasets. The results are within the expectation of what was claimed for the homophilic Planetoid datasets (Cora, Citeseer, and Pubmed) [18]. We also performed tests on real-world heterophilic data. The datasets from WebKB (Cornell, Texas, and Wisconsin) proved to be more difficult than originally claimed. When using the authors' code with their chosen hyperparameters the results are almost ten percentage points worse than what is claimed in the paper. This poor showing is further reflected in the actor dataset where a similar disparity can be found. Though our reproduction of the PolyGCL concept seems to perform more in line with what we would expect; Here all results fall within five percentage points of the claimed accuracy. The largest disparity, however, can be found when analyzing the WikipediaNetwork datasets (Chameleon and Squirrel). Here we consistently see a difference between claimed results and measured results larger than 10 percentage points, both with the authors' model and ours. We speculate that the cause of this divergence could be a data leakage in the version of these datasets used by the authors [16]. We do, however, see a significant improvement in our model when compared to the original authors.

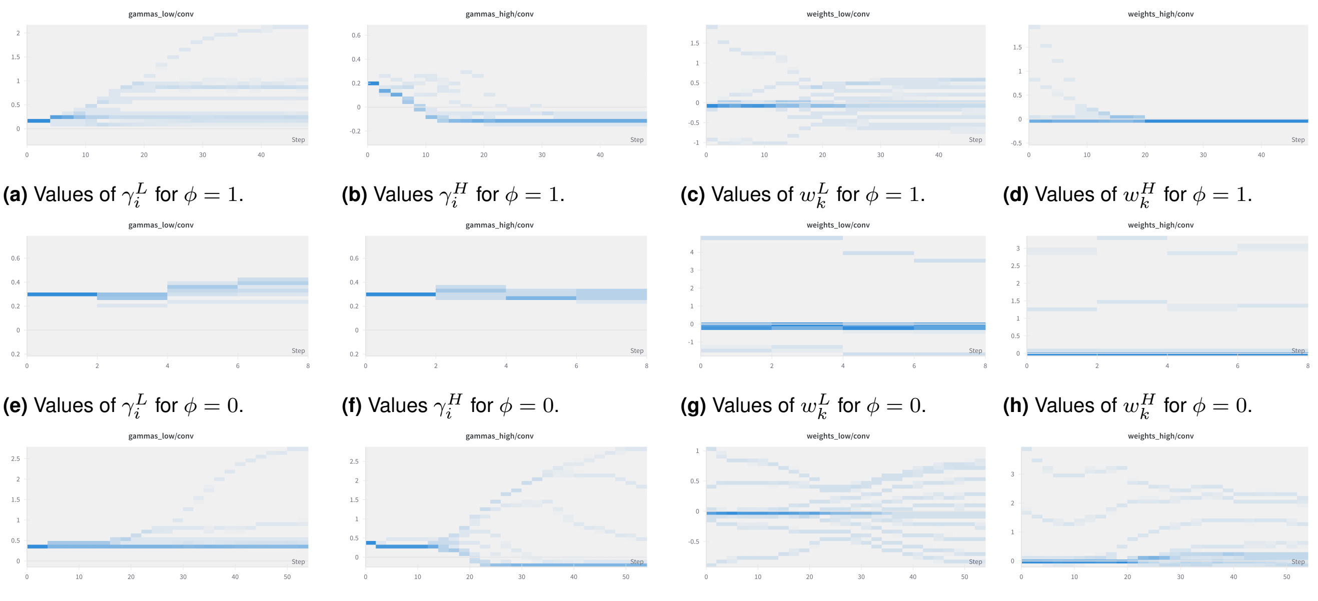
In Table 6, we can see further testing on real-world heterophilic datasets. For these datasets the results are all within an expected range of what was claimed, with both our model and the authors. This result is expected and reinforces the original authors' observation that PolyGCL, when compared to other methods, tends to perform better on heterophilic data.

7 Discussion and conclusion

Overall, PolyGCL proved to be an effective model for learning meaningful node representations in an unsupervised setting for both heterophilic and homophilic graph datasets. It introduces a novel approach to graph contrastive learning, built upon an encoder that produces representations considering two spectral views based on learnable low-pass and high-pass filters. Furthermore, the contribution of each view is weighted by another learnable parameter. We showed the importance of the low-pass and high-pass views on synthetic datasets, where the degree of homophily on the graphs can be controlled. Additionally, we showed that these views are learned correctly and the claimed accuracy can be reproduced with a simple linear classification as a downstream task that utilizes the learned representations. We further proved that on real-world datasets, it is for the most part possible to achieve the claimed results with the same simple linear classifier trained on the representation produced by PolyGCL. There were however two very problematic datasets that did not produce the expected results when PolyGCL was applied to them. Regarding this inconsistency, we were unable to identify a definitive explanation.

For the reproduction itself, we noted that by simply following the explanations of the paper, the model can underperform when compared to the expected results. Small but important details that were only clear when the code was open-sourced made an important difference in the effectiveness of the method.

One notable concern with the proposed method is the reliance on supervised metrics, such as the final accuracy of a linear classifier trained on the learned representations, to determine optimal hyperparameters of the encoder. In our experience, despite the model being designed to function in an unsupervised manner, the sensitivity of the model to hyperparameter settings necessitated the use of labeled data for fine-tuning of these parameters and reproduction the



(a) Values of γ_i^L for $\phi = 1$. **(b)** Values γ_i^H for $\phi = 1$. **(c)** Values of w_k^L for $\phi = 1$. **(d)** Values of w_k^H for $\phi = 1$.

(e) Values of γ_i^L for $\phi = 0$. **(f)** Values γ_i^H for $\phi = 0$. **(g)** Values of w_k^L for $\phi = 0$. **(h)** Values of w_k^H for $\phi = 0$.

(i) Values of γ_i^L for $\phi = -1$. **(j)** Values γ_i^H for $\phi = -1$. **(k)** Values of w_k^L for $\phi = -1$. **(l)** Values of w_k^H for $\phi = -1$.

Figure 2. Evolution of the values of γ_i and w_k for both the high pass and low pass view of the encoder.

Table 5. Results for homophilic and heterophilic datasets. The first row shows the results claimed in the original paper, then the results obtained by running the experiments with the publicly available code, and lastly the results obtained with our reproduction.

Methods	Cora	Citeseer	Pubmed	Cornell	Texas	Wisconsin	Actor	Chameleon	Squirrel
Claimed	$87.57_{\pm 0.62}$	$79.81_{\pm 0.85}$	$87.15_{\pm 0.27}$	$82.62_{\pm 3.11}$	$88.03_{\pm 1.80}$	$85.50_{\pm 1.88}$	$41.15_{\pm 0.88}$	$71.62_{\pm 0.96}$	$56.49_{\pm 0.72}$
Reproduced	$86.16_{\pm 0.84}$	$79.24_{\pm 0.64}$	$86.75_{\pm 0.26}$	$76.81_{\pm 3.62}$	$78.36_{\pm 3.28}$	$82.75_{\pm 2.38}$	$31.64_{\pm 0.68}$	$33.24_{\pm 1.25}$	$32.98_{\pm 0.71}$
Ours	$85.55_{\pm 0.62}$	$76.49_{\pm 0.98}$	$83.73_{\pm 0.26}$	$79.36_{\pm 3.41}$	$85.08_{\pm 1.48}$	$86.13_{\pm 2.00}$	$36.49_{\pm 0.61}$	$47.46_{\pm 0.94}$	$34.68_{\pm 0.77}$

Table 6. Results for heterophilic datasets. The first row shows the results claimed in the original paper, then the results obtained by running the experiments with the publicly available code, and lastly the results obtained with our reproduction.

Methods	Roman-empire	Amazon-ratings	Minesweeper (AUC)	Tolokers (AUC)	Questions (AUC)
Claimed	$72.97_{\pm 0.25}$	$44.29_{\pm 0.43}$	$86.11_{\pm 0.43}$	$83.73_{\pm 0.53}$	$75.33_{\pm 0.67}$
Reproduced	$72.44_{\pm 0.30}$	$43.83_{\pm 0.28}$	$86.12_{\pm 0.43}$	$83.70_{\pm 0.61}$	$74.87_{\pm 0.76}$
Ours	$68.36_{\pm 0.41}$	$48.84_{\pm 0.29}$	$85.03_{\pm 0.45}$	$83.80_{\pm 0.44}$	$68.11_{\pm 0.54}$

Table 7. Results for synthetic cSBM datasets. These datasets are constructed with a varying degree of homophily from very heterophilic to very homophilic. The first row shows the results claimed in the original paper, then the results obtained by running the experiments with the publicly available code, and lastly the results obtained with our reproduction.

ϕ	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1
Claimed	$98.84_{\pm 0.17}$	$94.23_{\pm 0.31}$	$90.82_{\pm 0.50}$	$75.43_{\pm 0.68}$	$66.51_{\pm 0.69}$	$69.43_{\pm 0.65}$	$88.22_{\pm 0.72}$	$98.09_{\pm 0.29}$	$99.29_{\pm 0.23}$
Reproduced	$88.99_{\pm 0.45}$	$93.49_{\pm 0.29}$	$90.41_{\pm 0.67}$	$74.94_{\pm 0.39}$	$67.25_{\pm 0.47}$	$70.17_{\pm 0.57}$	$88.39_{\pm 0.40}$	$97.68_{\pm 0.25}$	$98.64_{\pm 0.18}$
Ours	$98.34_{\pm 0.31}$	$98.48_{\pm 0.15}$	$91.18_{\pm 0.49}$	$70.80_{\pm 0.80}$	$63.99_{\pm 0.67}$	$64.99_{\pm 0.63}$	$91.06_{\pm 0.32}$	$97.70_{\pm 0.27}$	$99.22_{\pm 0.15}$

claimed results. Looking at the unsupervised loss alone did not always result in the best model choice. This introduces a contradiction, as in theory the model is supposed to learn in an unsupervised way.

8 Acknowledgments

We want to thank the authors of PolyGCL: Jingyu Chen, Runlin Lei, and Zhewei Wei for their work. In particular, we thank Jingyu Chen for the openness to dialogue, valuable feedback, and prompt responses throughout the preparation of this reproduction paper.

References

[1] Jingyu Chen, Runlin Lei, and Zhewei Wei. PolyGCL: GRAPH CONTRASTIVE LEARNING via learnable spectral polynomial filters. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=y21ZO6M86t`.

[2] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478, 2021. URL `https://arxiv.`

org/abs/2104.13478.

[3] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[4] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL http://arxiv.org/abs/1609.02907.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016. URL http://arxiv.org/abs/1606.09375.

[7] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory, 2009.

[8] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2014.

[9] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, May 2013. ISSN 1053-5888. doi: 10.1109/msp.2012.2235192. URL http://dx.doi.org/10.1109/MSP.2012.2235192.

[10] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep Graph Infomax. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rklz9iAcKQ.

[11] Mingguo He, Zhewei Wei, and Ji-Rong Wen. Convolutional neural networks on graphs with chebyshev approximation, revisited, 2024.

[12] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 35(2):2708–2718, February 2024. ISSN 2162-2388. doi: 10.1109/tnnls.2022.3190922. URL http://dx.doi.org/10.1109/TNNLS.2022.3190922.

[13] Mingguo He, Zhewei Wei, and Ji-Rong Wen. Convolutional neural networks on graphs with chebyshev approximation, revisited. In *NeurIPS*, 2022.

[14] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. In *NeurIPS*, 2021.

[15] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network, 2021.

[16] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at evaluation of gnns under heterophily: Are we really making progress? In *The Eleventh International Conference on Learning Representations*, 2023.

[17] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation, 2022.

[18] Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. 03 2016.

[19] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

[20] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *CoRR*, abs/1909.13021, 2019. URL http://arxiv.org/abs/1909.13021.

[21] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of gnns under heterophily: Are we really making progress?, 2024.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[23] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[24] Yash Deshpande, Andrea Montanari, Elchanan Mossel, and Subhabrata Sen. Contextual stochastic block models, 2018.