FiGURe: Simple and Efficient Unsupervised Node Representations with Filter Augmentations

Chanakya Ekbote*
Microsoft Research India
chanakyaekbote@gmail.com

Ajinkya Pankaj Deshpande*
Microsoft Research India
ajinkya.deshpande56@gmail.com

Arun Iyer
Microsoft Research India
ariy@microsoft.com

Ramakrishna Bairi
Microsoft Research India
rkbairi@gmail.com

Sundararajan Sellamanickam
Microsoft Research India
ssrajan@microsoft.com

ABSTRACT

Unsupervised node representations learnt using contrastive learning-based methods have shown good performance on downstream tasks. However, these methods rely on augmentations that mimic low-pass filters, limiting their performance on tasks requiring different eigen-spectrum parts. This paper presents a simple filter-based augmentation method to capture different parts of the eigen-spectrum. We show significant improvements using these augmentations. Further, we show that sharing the same weights across these different filter augmentations is possible, reducing the computational load. In addition, previous works have shown that good performance on downstream tasks requires high dimensional representations. Working with high dimensions increases the computations, especially when multiple augmentations are involved. We mitigate this problem and recover good performance through lower dimensional embeddings using simple random Fourier feature projections. Our method, FiGURe, achieves an average gain of up to 4.4%, compared to the state-of-the-art unsupervised models, across all datasets in consideration, both homophilic and heterophilic. Our code can be found here: https://github.com/microsoft/figure.

Paper Type: Evaluatory papers which revisit validity of domain assumptions, Work-in-progress papers.

CCS CONCEPTS

• **Computing methodologies** → **Dimensionality reduction and manifold learning**; **Neural networks**; **Kernel methods**.

KEYWORDS

graph neural networks, contrastive learning, kernel methods

*Both authors contributed equally to this work.

KDD MLG Workshop '23, August, 2023, Long Beach, California

ACM Reference Format:
Chanakya Ekbote, Ajinkya Pankaj Deshpande, Arun Iyer, Ramakrishna Bairi, and Sundararajan Sellamanickam. 2023. FiGURe: Simple and Efficient Unsupervised Node Representations with Filter Augmentations. In *Proceedings of KDD MLG Workshop '23.* ACM, New York, NY, USA, 14 pages.

1 INTRODUCTION

Contrastive learning is a powerful method for unsupervised graph representation learning, achieving notable success in various applications [11, 37]. However, these evaluations typically focus on tasks exhibiting homophily, where task labels strongly correlate with the graph's structure. An existing edge suggests the connected nodes likely share similar labels in these scenarios. However, these representations often struggle when dealing with heterophilic tasks, where edges tend to connect nodes with different labels.

Several papers [3, 6, 13, 23] have tackled the problem of heterophily by leveraging information from both low and high frequency components. However, these methods operate in the semi-supervised setting, and the extension of these ideas in unsupervised learning still needs to be explored. Inspired by the insights in these papers, we propose a simple method incorporating these principles. Our approach introduces filter banks as additional views and learns separate representations for each filter bank. However, this approach faces two main challenges: Firstly, storing representations from each view can become prohibitively expensive for large graphs; secondly, contrastive learning methods typically demand high-dimensional representations, which increase both the computational cost of training and the storage burden.

We employ a shared encoder for all filter banks to tackle the first challenge. Our results confirm that a shared encoder performs on par with independent encoders for each filter bank. This strategy enables us to reconstruct filter-specific representations as needed, drastically reducing the storage requirement.

For the second challenge, we train our models with low dimensional embeddings. Then, we use random Fourier feature projection [33] to lift these low-dimensional embeddings into a higher-dimensional space. Kernel tricks [18] were typically used in classical machine learning to project low-dimensional representation to high dimensions where the labels can become linearly separable. However, constructing and leveraging the kernels in large dataset scenarios could be expensive. To avoid this issue, several papers [16, 21, 30, 33, 34] proposed to approximate the map associated with the kernel. For our scenario, we use the map associated with Gaussian kernel [33]. We empirically demonstrate that using such a

simple approach preserves high performance for downstream tasks, even in the contrastive learning setting. Consequently, our solution offers a more efficient approach to unsupervised graph representation learning in computation and storage, especially concerning heterophilic tasks. The proposed method exhibits simplicity not only in the augmentation of filters but also in its ability to learn and capture information in a low-dimensional space, while still benefiting from the advantages of large-dimensional embeddings through Random Fourier Feature projections.

Our contributions in this work are, 1] We propose a simple scheme of using filter banks for learning representations that can cater to both heterophily and homophily tasks, 2] We address the computational and storage burden associated with this simple strategy by sharing the encoder across these various filter views, 3] By learning a low-dimensional representation and later projecting it to high dimensions using random Fourier Features, we further reduce the burden, 4] We study the performance of our approach on four homophilic and five heterophilic datasets. Our method, FiGURe, achieves an average gain of up to 4.4%, compared to the state-of-the-art unsupervised models, across all datasets in consideration, both homophilic and heterophilic. Notably, even without access to task-specific labels, FiGURe performs competitively with supervised methods like GCN [20].

2 RELATED WORK

Several unsupervised representation learning methods have been proposed in prior literature. Random walk-based methods like Node2Vec [10] and DeepWalk [31] preserve node proximity but tend to neglect structural information and node features. Contrastive methods, such as Deep Graph InfoMax (DGI) [37], maximize the mutual information (MI) between local and global representations while minimizing the MI between corrupted representations. Methods like MVGRL [11] and GRACE [38] expand on this, by integrating additional views into the MI maximization objective. However, most of these methods focus on the low frequency components, overlooking critical insights from other parts. Semi-supervised methods like GPRGNN [6], BernNet [13], and PPGNN [23] address this by exploring the entire eigenspectrum, but these concepts are yet to be applied in the unsupervised domain. This work proposes the use of a filter bank to capture information across the full eigenspectrum while sharing an encoder across filters. Given the high-dimensional representation demand of contrastive learning methods, we propose using Random Fourier Features (RFF) to project lower-dimensional embeddings into higher-dimensional spaces, reducing computational load without sacrificing performance. The ensuing sections define our problem, describe filter banks and random feature maps, and explain our model and experimental results.

3 PROBLEM SETTING

In the domain of unsupervised representation learning, our focus lies on graph data, denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ the set of edges ($\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$). We associate an adjacency matrix with $\mathcal{G}$, referred to as $\mathbf{A} : \mathbf{A} \in \{0,1\}^{n \times n}$, where $n = |\mathcal{V}|$ corresponds to the number of nodes. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the feature matrix. We use $\mathbf{A_I}$ to represent $\mathbf{A} + \mathbf{I}$ with $\mathbf{I}$ is the identity matrix, while $\mathbf{D}_{\mathbf{A_I}}$ signifies the degree matrix of $\mathbf{A_I}$. We also define $\mathbf{A_n}$ as $\mathbf{D}_{\mathbf{A_I}}^{-1/2}\mathbf{A_I}\mathbf{D}_{\mathbf{A_I}}^{-1/2}$. No additional information is provided during training. The goal is to learn a parameterized encoder, $E_\theta : \mathbb{R}^{n \times n} \times \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d'}$, where $d' \ll d$. This encoder produces a set of node representations $E_\theta(\mathbf{X}, \mathbf{A_n}) = \{h_1, h_2, ..., h_n\}$ where each $h_i \in \mathbb{R}^{d'}$ represents a rich representation for node i. The subsequent section will provide preliminary details about filter banks and random feature maps before we discuss the specifics of the proposed approach.

4 PRELIMINARIES

Our proposed approach hinges on the critical components of filter banks and random feature maps. In this section, we delve into brief details about these two facets, setting the stage for a comprehensive description of our approach.

4.1 Filter Banks

Graph Fourier Transform (GFT) forms the basis of Graph Neural Networks (GNNs). A GFT is defined using a reference operator $\mathbf{R}$ which admits a spectral decomposition. Traditionally, in the case of GNNs, this reference operator has been the symmetric normalized laplacian $\mathbf{L_n} = \mathbf{I} - \mathbf{A_n}$ or the $\mathbf{A_n}$ as simplified in [20]. A graph filter is an operator that acts independently on the entire eigenspace of a diagonalisable and symmetric reference operator $\mathbf{R}$, by modulating their corresponding eigenvalues. [35, 36]. Thus, a graph filter $\mathbf{H}$ is defined via the graph filter function $g(.)$ operating on the reference operator as $\mathbf{H} = g(\mathbf{R}) = \mathbf{U}g(\mathbf{\Lambda})\mathbf{U}^T$. Here, $\mathbf{\Lambda} = diag([\lambda_1, \lambda_2, ..., \lambda_n])$, where λ_i denotes the eigenvalues of the reference operator.

We describe a filter bank as a set of filters, denoted as $\mathbf{F} = \{\mathbf{F}_1, \mathbf{F}_2, ..., \mathbf{F}_K\}$. Both GPRGNN [6] and BernNet [13] employ filter banks, comprising polynomial filters, and amalgamate the representations from each filter bank to enhance the performance across heterophilic datasets. GPRGNN uses a filter bank defined as $\mathbf{F}_{\text{GPRGNN}} = \{\mathbf{I}, \mathbf{A_n}, ..., {\mathbf{A_n}}^{K-1}\}$, while $\mathbf{F}_{\text{BernNet}} = \{\mathbf{B}_0, \mathbf{B}_1, ..., \mathbf{B}_{K-1}\}$ characterizes the filter bank utilized by BernNet. Here, $\mathbf{B}_i = \frac{1}{2^{K-1}}\binom{K-1}{i}(2\mathbf{I} - \mathbf{L_n})^{K-i-1}(\mathbf{L_n})^i$.

Each filter in these banks highlights different parts of the eigenspectrum. By tuning the combination on downstream tasks, it offers the choice to select and leverage the right spectrum to enhance performance. Notably, unlike traditional GNNs, which primarily emphasize low-frequency components, higher frequency components have proved useful for heterophily [3, 6, 13, 23]. Consequently, a vital takeaway is that **for comprehensive representations, we must aggregate information from different parts of the eigenspectrum and fine-tune it for specific downstream tasks**.

4.2 Random Feature Maps for Kernel Approximations

Before the emergence of deep learning models, the kernel trick was instrumental in learning non-linear models. A kernel function, $k : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$, accepts two input features and returns a real-valued score. Given a positive-definite kernel, Mercer's Theorem [25] assures the existence of a feature map $\phi(\cdot)$, such that $k(x, y) = \langle \phi(x), \phi(y) \rangle$. Leveraging the kernel trick, researchers combined Mercer's theorem with the representer theorem [18], enabling the construction of non-linear models that remain linear in

k. These models created directly using k instead of the potentially complex ϕ, outperformed traditional linear models. The implicit maps linked with these kernels projected the features into a significantly high-dimensional space, where targets were presumed to be linearly separable. However, computational challenges arose when dealing with large datasets.

Addressing these issues, subsequent works [16, 30, 33, 34] introduced approximations of the map associated with individual kernels through random projections into higher-dimensional spaces ($\phi'(.)$). This approach ensures that $\langle \phi'(\mathbf{x}), \phi'(\mathbf{y}) \rangle \approx k(x, y)$. These random feature maps are inexpensive to compute and affirm that simple projections to higher-dimensional spaces can achieve linear separability. The critical insight is that **computationally efficient random feature maps exist, capable of projecting lower-dimensional representations into higher dimensions. These projections enhance the adaptability of these representations for downstream tasks. Random Fourier features (RFF) [33] provide a prime example of such techniques**.

5 PROPOSED APPROACH

The following section delineates the process of unsupervised representation learning. Post that, we give details on how the representations learned from each filter bank is used in downstream tasks using random feature maps.

5.1 Unsupervised Representation Learning

Our method: **Fi**lter-based **G**raph **U**nsupervised **Re**presentation Learning (FiGURe) builds on concepts introduced in [14, 37], extending the maximization of mutual information between node and global filter representations for each filter in the filter bank $\mathbf{F} = \{\mathbf{F}_1, \mathbf{F}_2, ...\mathbf{F}_K\}$. We construct an encoder for each filter to maximize the mutual information between the input data and encoder output. For the i^{th} filter, we learn an encoder, $E_\theta : \mathcal{X}_i \rightarrow \mathcal{X}'_i$, denoted by learnable parameters θ. In this context, $\mathcal{X}_i$ represents a set of examples, where each example $[\widehat{\mathbf{X}_{ij}}, \widehat{\mathbf{F}_{ij}}] \in \mathcal{X}_i$ consists of a filter $\mathbf{F}_i$, its corresponding nodes and node features drawn from an empirical probability distribution $\mathbb{P}_i$, which captures the joint distribution of features and node representations $[\mathbf{X}, \mathbf{F}_i]$. $\mathcal{X}_i$ defines the set of representations learnt by the encoder on utilizing feature information as well as topological information from the samples, sampled from the joint distribution $\mathbb{P}_i$. The goal, aligned with [14, 24, 37], is to identify θ that maximizes mutual information between $[\mathbf{X}, \mathbf{F}_i]$ and $E_\theta(\mathbf{X}, \mathbf{F}_i)$, or $\mathcal{I}_i([\mathbf{X}, \mathbf{F}_i], E_\theta(\mathbf{X}, \mathbf{F}_i))$. While exact mutual information (MI) computation is unfeasible due to unavailable exact data and learned representations distributions, we can estimate the MI using the Jensen-Shannon MI estimator [7, 27], defined as:

$$\mathcal{I}^{\text{JSD}}_{i,\theta,\omega}([\mathbf{X}, \mathbf{F}_i], E_\theta(\mathbf{X}, \mathbf{F}_i)) := \mathbb{E}_{\mathbb{P}_i}[-\text{sp}(T_{\theta,\omega}([\widehat{\mathbf{X}_{ij}}, \widehat{\mathbf{F}_{ij}}], E_\theta(\widehat{\mathbf{X}_{ij}}, \widehat{\mathbf{F}_{ij}}))] - \mathbb{E}_{\mathbb{P}_i \times \tilde{\mathbb{P}}_i}[\text{sp}(T_{\theta,\omega}([\widehat{\mathbf{X}_{ij}}, \widetilde{\mathbf{F}_{ij}}], E_\theta(\widehat{\mathbf{X}_{ij}}, \widehat{\mathbf{F}_{ij}}))] \quad (1)$$

Here, $T_\omega : \mathcal{X}_i \times \mathcal{X'}_i \rightarrow \mathbb{R}$ represents a discriminator function with learnable parameters ω. Note that $[\tilde{\mathbf{X}_{ij}}, \tilde{\mathbf{F}_{ij}}]$ is an input sampled from $\tilde{\mathbb{P}}_i$, which is a marginal of the joint distribution of the input data and the learned node representations. The function sp(.) corresponds to the softplus function [8]. Additionally, $T_{\theta,\omega} = D_w \circ (\mathcal{R}(E_\theta(\widehat{\mathbf{X}_{ij}}, \widehat{\mathbf{F}_{ij}})), E_\theta(\widehat{\mathbf{X}_{ij}}, \widehat{\mathbf{F}_{ij}}))$, where $\mathcal{R}$ denotes the readout function responsible for summarizing all node representations by aggregating and distilling information into a global filter representation.

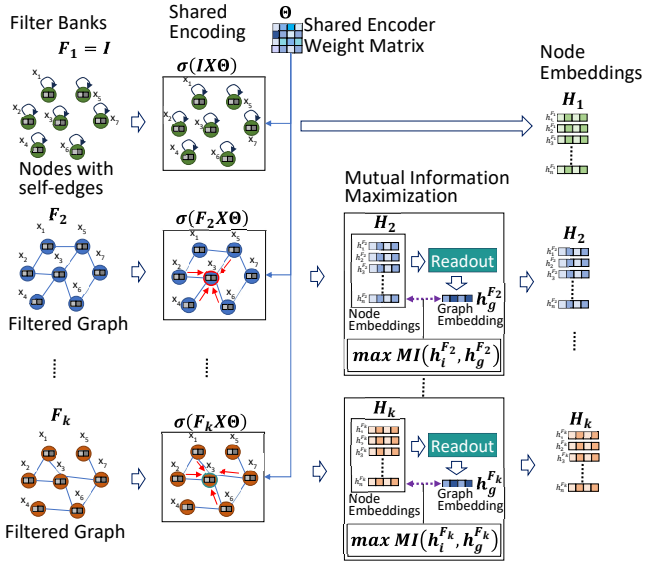

Figure 1: Unsupervised learning of node embeddings by maximizing mutual information between node and graph representations over the graphs from the filter bank. Note that the parameter Θ is shared across all the filters.

In our approach, we first obtain node representations by feeding the filter-specific topology and associated node features into the encoder: $\mathbf{H}_i = E_\theta(\mathbf{X}_i, \mathbf{F}_i) = \{h_1^{\mathbf{F}_i}, h_2^{\mathbf{F}_i}, ..., h_n^{\mathbf{F}_i}\}$. To obtain global representations, we employ a readout function $\mathcal{R} : \mathbb{R}^{N \times d'} \rightarrow \mathbb{R}^{d'}$, which combines and distills information into a global representation $h_g^{F_i} = \mathcal{R}(\mathbf{H}_i) = \mathcal{R}(E_\theta(\mathbf{X}, \mathbf{F}_i))$. Instead of directly maximizing the mutual information between the local and global representations, we introduce a learnable discriminator $D_\omega : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \rightarrow \mathbb{R}$, where $D_\omega(.,.)$ represents the joint probability score between the global representation and the node-specific patch representation. This joint probability score should be higher when considering global and local representations obtained from the same filter, as opposed to the joint probability score between the global representation from one filter and the local representation from an arbitrary filter.

To generate negative samples for contrastive learning, we employ a corruption function $C : \mathbb{R}^{N \times d} \times \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{M \times d} \times \mathbb{R}^{M \times M}$, which yields corrupted samples denoted as $[\widetilde{\mathbf{X}_{ij}}, \widetilde{\mathbf{F}_{ij}}] = C(\mathbf{X}, \mathbf{F}_i)$. The designed corruption function generates data decorrelated with the input data.

In order to learn representations across all filters in the filter bank, we aim to maximise the average estimate of mutual information (MI) across all filters, considering K filters.

$$\mathcal{I}_{\mathbf{F}} = \frac{1}{K} \sum_{i=1}^{K} \mathcal{I}^{JSD}_{i,\theta,\omega}([\mathbf{X}, \mathbf{F}_i], E_\theta(\mathbf{X}, \mathbf{F}_i)) \quad (2)$$

Maximising the Jenson-Shannon MI estimator is equivalent to reducing the binary cross entropy loss defined between positive samples (sampled from the joint) and the negative samples (sampled from the product of marginals). Therefore, for each filter, we minimise the following objective:

$$\mathcal{L}_{\mathbf{F}_i} = \frac{1}{N+M}\sum_{j=1}^{N}\mathbb{E}_{(\mathbf{X},\mathbf{F}_i)}[\log(D_\omega(h_j^{\mathbf{F}_i}, h_g^{\mathbf{F}_i}))] + \frac{1}{N+M}\sum_{j=1}^{M}\mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{F}_i})}[\log(1-D_\omega(\widetilde{h_j^{\mathbf{F}_i}}, h_g^{\mathbf{F}_i}))] \tag{3}$$

Therefore to learn meaningful representations across all filters the following objective is minimised:

$$\mathcal{L} = \frac{1}{K}\sum_{i=1}^{K}\mathcal{L}_{\mathbf{F}_i} \tag{4}$$

However, managing the computational cost of training and storage for large graphs with separate node representations for each filter presents a significant challenge, exacerbated by the high dimensional requirements of contrastive learning methods. We implement parameter sharing to mitigate the first issue, borrowing the concept from studies such as [6, 13], thereby sharing the encoder's parameters θ and the discriminator's parameters ω across all filters. Instead of storing dense filter-specific node representations, we only store the parameters of the shared encoder and the first-hop neighbourhood information of each node per filter, which has a lower storage cost. For downstream tasks, we retrieve the embeddings by reconstructing filter-specific representations. To ensure quick and efficient reconstruction, we use a simple one-layer GNN. This on-demand reconstruction of filter-specific representations significantly reduces the computational and storage requirements associated with individual node representations. Fig 1 illustrates such a simple encoder's mutual information-based learning process.

Addressing the second issue, we initially train our models to generate low-dimensional embeddings. These encapsulate latent classes, as discussed in [2] as a superset of classes pertinent to downstream tasks. Although the low-dimensional embeddings harbour latent class information, they lack linear separability. Hence, we project these embeddings into a higher-dimensional space using random Fourier feature (RFF) projections, a strategy inspired by kernel methods (Section 4.2). Using this approach allows for improved linear separability of the latent classes. Our experimental findings (Section 6.2) affirm the effectiveness of projecting lower-dimensional embeddings into higher dimensions, confirming the retention of latent class information in these embeddings.

5.2 Supervised Representation Learning

After obtaining representations for each filter post the reconstruction of the node representations, learning an aggregation mechanism to combine information from representations that capture different parts of the eigenspectrum for the given task is necessary. We adopt learning schemes proposed in [6, 13, 23], where we learn a weighted combination of filter-specific representations. Therefore, the combined representations we learn for the downstream task are as follows (considering K filters from the filter bank $\mathbf{F}$):

$$Z = \sum_{i=1}^{K}\alpha_i\phi'(E_\theta(\mathbf{X},\mathbf{F}_i)) \tag{5}$$

The parameters α_i's are learnable. Additionally, the function $\phi(.)'$ represents either the RFF projection or an identity transformation, depending on whether $E_\theta(\mathbf{X},\mathbf{F}_i)$ is low-dimensional or not. A classifier model (e.g. logistic regression) consumes these embeddings, where we train both the α_i's and the weights of the classifier. Fig 2 illustrates this process. The main distinction between semi-supervised methods such as [6, 13, 23] and our method is that the semi-supervised methods learn both the encoder and the combination coefficients based on labelled data. However, we pre-train the encoder in our method and subsequently learn a task-specific combination of filter-specific representations.

6 EXPERIMENTAL RESULTS

Training Details: We define a single-layer graph convolutional network (GCN) with shared weights (Θ) across all filters in the filter bank ($\mathbf{F}$) as our encoder. Therefore, the encoder can be expressed as follows: $E_\theta(\mathbf{X},\mathbf{F}_i) = \sigma(\mathbf{F}_i\mathbf{X}\Theta)$. It is important to note that $\mathbf{F}_i$ represents a normalized filter with self-loops, which ensures that its eigenvalues are within the range of $[0, 2]$. The non-linearity function σ refers to the parametric rectified linear unit (PReLU) [12]. As we work with a single graph, we obtain the positive samples by sampling nodes from the graph. Using these sampled nodes, we construct a new adjacency list that only includes the edges between these sampled nodes in filter $\mathbf{F}_i$. On the other hand, the corruption function C operates on the same sampled nodes. However, it randomly shuffles the node features instead of perturbing the adjacency list. Similar to [37], we employ a straightforward readout function that involves averaging the representations across all nodes for a specific filter $\mathbf{F}_i$: $\mathcal{R}(\mathbf{H}_i) = \sigma\left(\frac{1}{N}\sum_{j=0}^{N}h_j^{\mathbf{F}_i}\right)$ where σ denotes the sigmoid non-linearity. We utilize a bilinear scoring function, whose parameters are also shared across all filters:

$$D_\omega(h_j^{\mathbf{F}_i}, h_g^{\mathbf{F}_i}) = \sigma(h_j^{\mathbf{F}_i T}\mathbf{W}h_g^{\mathbf{F}_i}) \tag{6}$$

We learn the encoder and discriminator parameters by optimising Eq. 4. While we could use various filter banks, we specifically employ the filter bank corresponding to GPRGNN ($\mathbf{F}_{\text{GPRGNN}}$) for all our experiments. However, we also conduct an ablation study (see 6.6) to compare the performance when using $\mathbf{F}_{\text{GPRGNN}}$ versus $\mathbf{F}_{\text{BernNet}}$. For more detailed training information, please refer to the supplementary material.

We conducted a series of comprehensive experiments to evaluate the effectiveness and competitiveness of our proposed model compared to SOTA models and methods. These experiments address the following research questions: **[RQ1]** How does FiGURe, perform compared to SOTA unsupervised models? **[RQ2]** Can we perform satisfactorily even with lower dimensional representations using projections such as RFF? **[RQ3]** Does shared encoder decrease performance? **[RQ4]** What is the computational efficiency gained by using lower dimensional representations compared to graph contrastive methods that rely on higher dimensional representations? **[RQ5]** What is the computational efficiency gained by using lower dimensional representations compared to node contrastive

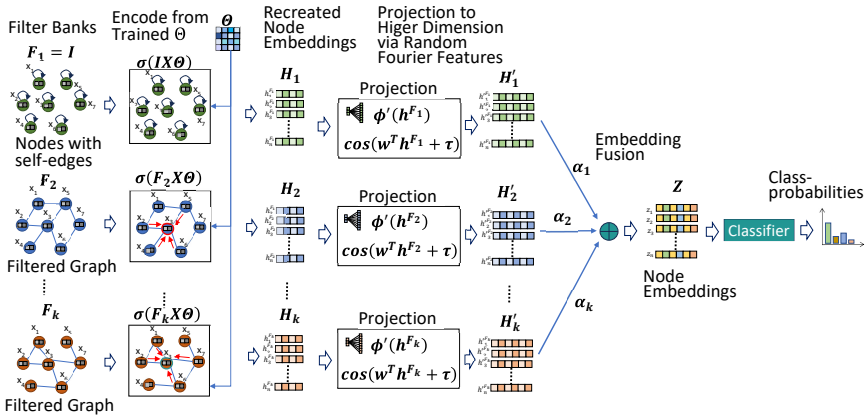

Figure 2: Supervised Learning: Using the trained parameter Θ, we generate the node embeddings by encoding the filtered graphs that get consumed in the classification task.

methods that rely on higher dimensional representations? **[RQ6]** Can alternative filter banks be employed to recover good quality representations?

Datasets and Setup: We evaluated our model on a diverse set of real-world datasets, which include both heterophilic and homophilic networks, to assess its effectiveness. Similar to previous works, we utilized the node classification task as a proxy to evaluate the quality of the learned representations. Please refer to the supplementary material for detailed information about the benchmark datasets.

The heterophilic datasets used in our evaluation include: **chameleon**, **squirrel**, **roman-empire**, **minesweeper** and **arXiv-Year**. For chameleon and squirrel, we adopted the ten random splits (with 48%, 32%, and 20% of nodes allocated for the train, validation, and test sets, respectively) from [29]. For roman-empire and minesweeper, we used the ten random splits provided in [32]. Additionally, we evaluated our model on four homophilic datasets: **cora**, **citeseer**, and **pubmed**, **OGBN-arXiv**, as borrowed from [29]. We report the mean and standard deviation of the test accuracy across different splits. Please refer to the supplementary material for detailed statistics of each dataset.

Baselines: In our comparison against baselines, we considered common unsupervised approaches, such as DeepWalk and Node2Vec, and state-of-the-art mutual information-based methods, namely DGI, MVGRL, GRACE, and SUGRL. We also include the performance numbers of the widely used GCNfor reference. It is important to note that unless explicitly mentioned, we set the representation size to 512 dimensions for all reported results, consistent with previous work. Please refer to the supplementary material for detailed comparisons with other supervised methods and the link to our codebase.

6.1 RQ1: FiGURe versus SOTA Methods

We analyzed the results in Table 1 and made important observations. Across homophilic and heterophilic datasets, FiGURe consistently outperforms several SOTA unsupervised models, except in a few cases where it achieves comparable performance. Even on the large-scale datasets arXiv-Year and OGBN-arXiv FiGURe performs well, demonstrating the scalability of our method. Two baseline methods MVGRL and GRACE run into memory issues on the larger datasets and are accordingly reported OOM in the table.

We want to emphasize the rightmost column of the table, which shows the average percentage gain in performance across all datasets. This metric compares the improvement that FiGURe provides over each baseline model for each dataset and averages these improvements. This metric highlights that FiGURe performs consistently well across diverse datasets. No other baseline model achieves the same consistent performance across all datasets as FiGURe. Even the recent state-of-the-art contrastive models GRACE and SUGRL experience average performance drops of approximately 5% and 10%, respectively. This result indicates that FiGURe learns representations that exhibit high generalization and task-agnostic capabilities. Another important observation is the effectiveness of RFF projections in improving lower dimensional representations. We compared FiGURe at different dimensions, including Fi$GURe_{32}$ and Fi$GURe_{128}$, corresponding to learning 32 and 128-dimensional embeddings, respectively, in addition to the baseline representation size of 512 dimensions. Remarkably, even at lower dimensions, FiGURe with RFF projections demonstrates competitive performance across datasets, surpassing the 512-dimensional baselines in several cases. This result highlights the effectiveness of RFF projections in enhancing the quality of lower dimensional representations. Using lower-dimensional embeddings reduces the computation time and makes FiGURe faster than the baselines. It is noteworthy that the computational efficiency gained by reducing the dimension size becomes significant with the scale of the dataset. On arXiv-Year, which is a large graph containing 169,343 nodes, 128 dimensional embeddings give ∼ 1.6x speedup and 32 dimensional embeddings give ∼ 1.7x speedup. A similar observation is made with OGBN-arXiv. Section 6.2 discusses more insights about the effectiveness of RFF projections, and Sections 6.4 and 6.5 shed more light on the computational efficiency gain.

Furthermore, we include the widely used supervised model, GCN, in Table 2 as a benchmark for comparison. Notably, FiGURe remains competitive on most datasets, in some cases even outperforming

Table 1: Contains node classification accuracy percentages on homophilic and heterophilic datasets. FiGURe$_{32}$ and FiGURe$_{128}$ refer to FiGURe trained with 32 and 128 dimensional representations, respectively, and then projected using RFF. The remaining models are trained at 512 dimensions. Higher numbers indicate better performance. It is worth noting that FiGURe achieves superior performance or remains competitive with the baseline methods in all cases. The rightmost column Av. Δ_{gain} represents the average accuracy % gain of FiGURe over the model in that row, averaged across the different datasets. In case one of the models is 'OOM' for a particular dataset, that cell is ignored while calculating the Av. Δ_{gain}. Blue, Red and Green represent the 1st, 2nd and 3rd best performing models, for a particular dataset.

	Heterophilic Datasets					Homophilic Datasets				
	squirrel	chameleon	roman-empire	minesweeper	arXiv-Year	cora	citeseer	pubmed	OGBN-arXiv	Av. Δ_{gain}
DeepWalk	38.66 (1.44)	53.42 (1.73)	13.08 (0.59)	79.96 (0.08)	41.05 (0.10)	83.64 (1.85)	63.66 (3.36)	80.85 (0.44)	64.02	13.48
Node2Vec	42.60 (1.15)	54.23 (2.30)	12.12 (0.30)	80.00 (0.00)	39.69 (0.09)	78.19 (1.14)	57.45 (6.44)	73.24 (0.59)	60.20	15.78
DGI	39.61 (1.81)	59.28 (1.23)	47.54 (0.76)	82.51 (0.47)	40.59 (0.09)	84.57 (1.22)	73.96 (1.61)	86.57 (0.52)	65.58	6.61
MVGRL	39.90 (1.39)	54.61 (2.29)	68.50 (0.38)	85.60 (0.35)	OOM	86.22 (1.30)	75.02 (1.72)	87.12 (0.35)	OOM	4.39
GRACE	53.15 (1.10)	68.25 (1.77)	47.83 (0.53)	80.22 (0.45)	OOM	84.79 (1.51)	67.60 (2.01)	87.04 (0.43)	OOM	5.54
SUGRL	43.13 (1.36)	58.60 (2.04)	39.40 (0.49)	82.40 (0.58)	36.96 (0.19)	81.21 (2.07)	67.50 (1.62)	86.90 (0.54)	65.80	8.64
FiGURe$_{32}$	48.89 (1.55)	65.66 (2.52)	64.61 (0.92)	85.28 (0.71)	41.30 (0.21)	82.56 (0.87)	71.25 (2.20)	83.91 (0.69)	66.58	3.65
FiGURe$_{128}$	48.78 (2.48)	66.03 (2.19)	67.01 (0.56)	85.16 (0.58)	41.94 (0.15)	86.14 (1.13)	73.34 (1.91)	83.56 (0.34)	69.11	2.53
FiGURe	52.23 (1.19)	68.55 (1.87)	70.99 (0.52)	85.58 (0.49)	42.26 (0.20)	87.00 (1.24)	74.77 (2.00)	88.60 (0.44)	69.69	0.00

Table 2: Comparison of Node classification accuracy percentages with the widely used supervised model GCN. Despite not having access to task specific labels, FiGURe learns good quality representations.

	squirrel	chameleon	roman-empire	minesweeper	arXiv-Year	cora	citeseer	pubmed	OGBN-arXiv
GCN	47.78 (2.13)	61.43 (2.70)	**73.69 (0.74)**	**89.75 (0.52)**	**46.02 (0.26)**	**87.36 (0.91)**	**76.47 (1.34)**	88.41 (0.46)	69.37 (0.00)
FiGURe	**52.23 (1.19)**	**68.55 (1.87)**	70.99 (0.52)	85.58 (0.49)	42.26 (0.20)	87.00 (1.24)	74.77 (2.00)	**88.60 (0.44)**	**69.69 (0.00)**

Table 3: Mean epoch time (in seconds) on the two large datasets for different embedding sizes. For lower dimensional embeddings, there is a significant speedup.

.	512 dims	128 dims	32 dims
Arxiv-year	1.24s	0.75s	0.72s
Ogbn-arxiv	0.92s	0.74s	0.72s

GCN. This demonstrates that the specific information that is required by the downstream task, captured by GCN, can be extracted using an unsupervised method like FiGURe. For a downstream task, utilizing embeddings coming from FiGURe, instead of using them directly, means that a much more computationally efficient model like Logistic Regression may be used, as opposed to training an end-to-end graph neural network which is known to be expensive. There are, however, works such as [5],[9] and [4] that are exploring how to speedup the end-to-end graph neural network training as well.

Thus, for a downstream task, FiGURe is a lot more computationally efficient than an end-to-end supervised model like a graph neural network, which is known to be computationally expensive. It is possible for the performance FiGURe to improve even further using a non-linear model like an MLP. Please refer to supplementary material for detailed comparisons with other supervised methods.

An interesting point to note is that both OGBN-arXiv and arXiv-Year use the arXiv citation network as the graph, however differ with respect to the labels. In OGBN-arXiv, the task is to predict the subject area and in arXiv-Year the task is to predict the year of publication. FiGURe manages to provide improvements in both cases, demonstrating task-agnostic, and therefore multi-task properties. Owing to the multiple filters utilized by FiGURe, the model learns a very general representation of the nodes, which allows diverse tasks to choose the information most beneficial to them (see Section 5.2) and leads to good performance across the board.

6.2 RQ2: RFF Projections on Lower Dimensional Representations

In this section, we analyse the performance of unsupervised baselines using 32-dimensional embeddings with and without RFF projections (see Table 4). Despite extensive hyperparameter tuning, we could not replicate the results reported by SUGRL, so we present the best results we obtained. Two noteworthy observations emerge from these tables. Firstly, it is evident that lower dimensional embeddings can yield meaningful and linearly separable representations when combined with simple RFF projections. Utilising RFF projections enhances performance in almost all cases, highlighting the value captured by MI-based methods even with lower-dimensional embeddings. Secondly, FiGURe consistently achieves superior or comparable performance to the baselines, even in lower dimensions. Notably, this includes SUGRL, purported to excel in such settings. However, there is a 2-3% performance gap between GRACE and our method for the squirrel and chameleon datasets. While GRACE handles heterophily well at lower dimensions, its performance deteriorates with homophilic graphs, unlike FiGURe which captures lower frequency information effectively. Additionally, our

Table 4: Node classification accuracy percentages with and without using Random Fourier Feature projections (on 32 dimensions). A higher number means better performance. The performance is improved by using RFF in almost all cases, indicating the usefulness of this transformation

	RFF	CORA	CITESEER	SQUIRREL	CHAMELEON
DGI	×	**81.65 (1.90)**	65.62 (2.39)	31.60 (2.19)	45.48 (3.02)
	✓	81.49 (1.96)	**66.50 (2.44)**	**38.19 (1.52)**	**56.01 (2.66)**
MVGRL	×	**81.03 (1.29)**	72.38 (1.68)	37.20 (1.22)	49.65 (2.08)
	✓	80.48 (1.71)	**72.54 (1.89)**	**39.53 (1.04)**	**56.73 (2.52)**
SUGRL	×	65.35 (2.41)	42.84 (2.57)	31.62 (1.47)	43.20 (1.79)
	✓	**70.06** (1.24)	**47.03** (3.02)	**38.50 (2.19)**	**51.01 (2.26)**
GRACE	×	76.84 (1.09)	58.40 (3.05)	38.20 (1.38)	53.25 (1.58)
	✓	**79.15 (1.44)**	**63.66 (2.96)**	**51.56 (1.39)**	**67.39 (2.23)**
FiGURe	×	**82.88 (1.42)**	70.32 (1.98)	39.38 (1.35)	53.27 (2.40)
	✓	82.56 (0.87)	**71.25 (2.20)**	**48.89 (1.55)**	**65.66 (2.52)**

method exhibits computational efficiency advantages for specific datasets in lower dimensions. Please refer to the supplementary material for more details. Overall, these findings highlight the potential of RFF projections in extracting useful information from lower dimensional embeddings and reaffirm the competitiveness of FiGURe over the baselines.

6.3 RQ3: Sharing Weights Across Filter Specific Encoders

Table 5: A comparison of the performance on the downstream node classification task using independently trained encoders and weight sharing across encoders is shown. The reported metric is accuracy. In both cases, the embeddings are combined using the method described in 5.2

	CORA	CITESEER	SQUIRREL	CHAMELEON
INDEPENDENT	86.92 (1.10) %	75.03 (1.75) %	50.52 (1.51) %	66.86 (1.85) %
SHARED	87.00 (1.24) %	74.77 (2.00) %	52.23 (1.19) %	68.55 (1.87) %

Our method proposes to reduce the computational load by sharing the encoder weights across all filters. It stands to reason whether sharing these weights causes any degradation in performance. We present the results with shared and independent encoders across the filters in Table 5 to verify this.

We hypothesize that weight sharing among encoders results in embedding different filter representations in a shared subspace, thereby enhancing their suitability for learning a combined representation. This ultimately leads to improved features for downstream tasks and, in some cases, results in performance improvements. The experimental results demonstrate that there is no significant decrease in performance when using shared weights and, in fact, in some cases, performance is enhanced. This validates the claim that shared encoders can effectively reduce computational load without sacrificing performance.

6.4 RQ4: Computational Efficiency - Graph Contrastive

Table 6: Mean epoch time (in milliseconds) averaged across 20 trials with different hyperparameters. A lower number means the method is faster. Even though our method is slower at 512 dimensions, using 128 and 32 dimensional embeddings significantly reduces the mean epoch time. Using RFF as described in 6.2 we are able to prevent the performance drops experienced by DGI and MVGRL.

	DGI	MVGRL	FiGURe	FiGURe$_{128}$	FiGURe$_{32}$
CORA	38.53 (0.77)	75.29 (0.56)	114.38 (0.51)	20.10 (0.46)	11.54 (0.34)
CITESEER	52.98 (1.15)	102.41 (0.99)	156.24 (0.56)	30.30 (0.60)	17.16 (0.51)
SQUIRREL	87.06 (2.07)	168.24 (2.08)	257.65 (0.76)	47.72 (1.40)	23.52 (1.14)
CHAMELEON	33.08 (0.49)	64.71 (1.05)	98.36 (0.64)	18.56 (0.39)	11.63 (0.48)

There are two broad types of unsupervised methods for graph datasets: Graph Contrastive (GC) methods and Node Contrastive (NC) methods. GC methods, including MVGRL, DGI and FiGURe belong to the category of unsupervised methods that perform contrastive learning with representations of the entire graph. On the other hand, NC methods, such as SUGRL and GRACE, fall into a different class where they contrast against other node representations without the need for graph representations. In this section, we focus on comparing the computational efficiency of FiGURe with other GC methods, while in the next section, we compare its computational efficiency with NC methods. Hence, to assess the computational efficiency of the different GC methods, we analyzed the computation time and summarized the results in Table 6. The key metric used in this analysis is the mean epoch time: the average time taken to complete one epoch of training. We compared our method with other GC based methods such as DGI and MVGRL. Due to the increase in the number of augmentation views, there is an expected increase in computation time from DGI to MVGRL to FiGURe. However, as demonstrated in 6.2, using RFF projections allows us to achieve competitive performance even at lower dimensions. Therefore, we also included comparisons with our method at 128 and 32 dimensions in the table.

It is evident from the results that our method, both at 128 and 32 dimensions, exhibits faster computation times compared to both DGI and MVGRL, which rely on higher-dimensional representations to achieve good performance. This result indicates that FiGURe is computationally efficient due to its ability to work with lower-dimensional representations. During training, our method, F*i*GURe$_{32}$, is $\sim$ 3x faster than DGI and $\sim$ 6x times faster than MVGRL. Despite the faster computation, F*i*GURe$_{32}$ also exhibits an average performance improvement of around 2% across the datasets over all methods considered in our experiments. Therefore, among all GC methods, FiGURe with RFF not only achieves better performance but also demonstrates higher computational efficiency.

Table 7: Mean epoch time (in milliseconds) averaged across 20 trials with different hyperparameters. A lower number means the method is faster. Even though our method is slower at 512 dimensions, using 128 and 32 dimensional embeddings significantly reduces the mean epoch time. Using RFF as described in 6.2 we are able to prevent the performance drops experienced by SUGRL and GRACE.

	SUGRL	GRACE	FiGURe	FiGURe$_{128}$	FiGURe$_{32}$
CORA	15.92 (4.10)	51.19 (6.8)	114.38 (0.51)	20.10 (0.46)	11.54 (0.34)
CITESEER	24.37 (4.92)	77.16 (7.2)	156.24 (0.56)	30.30 (0.60)	17.16 (0.51)
SQUIRREL	33.63 (6.94)	355.2 (67.34)	257.65 (0.76)	47.72 (1.40)	23.52 (1.14)
CHAMELEON	16.91 (5.90)	85.05 (14.1)	98.36 (0.64)	18.56 (0.39)	11.63 (0.48)

6.5 RQ5: Computational Efficiency - Node Contrastive

In this section, we compare the computational efficiency of FiGURe with other NC methods. It is worth noting that NC methods do not require the computation of the graph representation, which leads to higher computational efficiency for these methods. Hence, as upon initial inspection of Table 7, it appears that SUGRL (at 512 dimensions) exhibits the highest computational efficiency, even outperforming F*i*GURe$_{128}$. However, despite its computational efficiency, the significant drop in performance across datasets (as discussed in Section 6.1) renders it less favorable for consideration. In fact, F*i*GURe$_{32}$ offers computational cost savings compared to SUGRL, while also achieving significantly better downstream classification accuracy. Turning to GRACE, it demonstrates greater computational efficiency than FiGURe (at 512 dimensions) for low to medium-sized graphs. However, as the graph size increases, due to random node feature level masking and edge level masking, the computational requirements of GRACE substantially increase (as evidenced by the results on SQUIRREL). Therefore, for larger graphs with more than approximately 5000 nodes, FiGURe proves to be more computationally efficient than GRACE (even at 512 dimensions). Furthermore, considering the performance improvements exhibited by FiGURe, it is evident that FiGURe (combined with RFF projections) emerges as the preferred method for unsupervised contrastive learning in graph data.

An interesting research direction would involve incorporating NC methods with filters, which would allow us to benefit from the performance improvements provided by filters while also achieving better computational efficiency. However, exploring this direction is beyond the scope of this paper, and we consider it as a potential avenue for future work.

6.6 RQ6: Experiments on Other Filter Banks

To showcase the versatility of our proposed framework, we conducted an experiment using Bernstein filters, as detailed in Table 8. The results indicate that using $\mathbf{F}_{\text{GPRGNN}}$ leads to better performance than Bernstein filters. We believe that the reason this is happening is due to the latent characteristics of the dataset. [13, 23] have shown that datasets like CHAMELEON and SQUIRREL need frequency response functions that give more prominence to the tail-end spectrum. $\mathbf{F}_{\text{GPRGNN}}$ are more amenable to these needs, as demonstrated in [23]. However, datasets requiring frequency response similar to comb filters may be better approximated by $\mathbf{F}_{\text{BernNet}}$ as their basis gives uniform prominence on the entire spectrum. Please refer to the supplementary material, which shows the basis frequency responses of these two filter banks, with more clarification. Therefore, although $\mathbf{F}_{\text{GPRGNN}}$ gives better performance for these datasets, there could be datasets where $\mathbf{F}_{\text{BernNet}}$ could do better. Hence, we proposed a general framework that can work with any filter bank.

Table 8: Accuracy percentage results using other filter banks for FiGURe. $\mathbf{F}^{3}_{\text{BernNet}}$ refers to the $\mathbf{F}_{\text{BernNet}}$ filter bank (Section 4.1) with K set to 3 and $\mathbf{F}^{11}_{\text{BernNet}}$ refers to K set to 11.

	CORA	CITESEER	SQUIRREL	CHAMELEON
$\mathbf{F}^{3}_{\text{BernNet}}$	85.13 (1.26)	73.38 (1.81)	37.07 (1.29)	53.95 (2.78)
$\mathbf{F}^{11}_{\text{BernNet}}$	86.62 (1.59)	73.97 (1.43)	43.48 (3.80)	62.13 (3.66)
$\mathbf{F}_{\text{GPRGNN}}$	87.00 (1.24)	74.77 (2.00)	52.23 (1.19)	68.55 (1.87)

7 CONCLUSION AND FUTURE WORK

Our work demonstrates the benefits of enhancing contrastive learning methods with filter views and learning filter-specific representations to cater to diverse tasks from homophily to heterophily. We have effectively alleviated computational and storage burdens by sharing the encoder across these filters and focusing on low-dimensional embeddings that utilize high-dimensional projections, a technique inspired by random feature maps developed for kernel approximations. Future directions include extending the analysis in [2] to graph contrastive learning and explicitly exploring the linear separability in low dimensions. This analysis could solidify the connection with the proposed random feature maps approach. Another future direction worth exploring is the combination of filters with Node Contrastive (NC) methods. By incorporating filters into NC methods, we can potentially leverage the performance benefits of filters while achieving improved computational efficiency. This integration of filters and NC methods could lead to more effective and scalable unsupervised learning approaches for graph data.

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *International conference on knowledge discovery & data mining (KDD)*. 2623–2631.

[2] Sanjeev Arora, Hrishikesh Khandeparkar, Mikhail Khodak, Orestis Plevrakis, and Nikunj Saunshi. 2019. A Theoretical Analysis of Contrastive Unsupervised Representation Learning. In *International Conference on Machine Learning (ICML)*.

[3] Deyu Bo, X. Wang, Chuan Shi, and Hua-Wei Shen. 2021. Beyond Low-frequency Information in Graph Convolutional Networks. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

[4] Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 2464–2473. https://doi.org/10.1145/3394486.3403296

[5] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rytstxWAW

[6] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *International Conference on Learning Representations (ICLR)*.

[7] Monroe D. Donsker and S. R. S. Varadhan. 1975. Asymptotic evaluation of certain Markov process expectations for large time. In *Communications on Pure and*

Applied Mathematics.

[8] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. 2000. Incorporating Second-Order Functional Knowledge for Better Option Pricing. In *Neural Information Processing Systems (NeurIPS)*. 7 pages.

[9] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. In *ICML Workshop on Graph Representation Learning and Beyond*. https://arxiv.org/abs/2004.11198

[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *International Conference on Knowledge Discovery and Data Mining (KDD)*.

[11] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive Multi-View Representation Learning on Graphs. In *International Conference on Machine Learning (ICML)*.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*. https://doi.org/10.1109/ICCV.2015.123

[13] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. 2022. BernNet: Learning Arbitrary Graph Spectral Filters via Bernstein Approximation. In *Neural Information Processing Systems (NeurIPS)*.

[14] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2019. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=Bklr3j0cKX

[15] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *Neural Information Processing Systems (NeurIPS)*.

[16] Purushottam Kar and Harish Karnick. 2012. Random Feature Maps for Dot Product Kernels. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. https://proceedings.mlr.press/v22/kar12.html

[17] Dongkwan Kim and Alice Oh. 2021. How to Find Your Friendly Neighborhood: Graph Attention Design with Self-Supervision. In *International Conference on Learning Representations (ICLR)*.

[18] George Kimeldorf and Grace Wahba. 1971. Some results on Tchebycheffian spline functions. *J. Math. Anal. Appl.* (1971). https://doi.org/10.1016/0022-247X(71)90184-3

[19] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

[20] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[21] Zhu Li, Jean-Francois Ton, Dino Oglic, and Dino Sejdinovic. 2021. Towards a Unified Analysis of Random Fourier Features. *Journal of Machine Learning Research (JMLR)* (2021).

[22] Derek Lim, Felix Matthew Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Prasad Bhalerao, and Ser-Nam Lim. 2021. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. In *Neural Information Processing Systems (NeurIPS)*. https://openreview.net/forum?id=DfGu8WwT0d

[23] Vijay Lingam, Chanakya Ekbote, Manan Sharma, Rahul Ragesh, Arun Iyer, and Sundararajan Sellamanickam. 2022. A Piece-wise Polynomial Filtering Approach for Graph Neural Networks. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*.

[24] R. Linsker. 1988. Self-organization in a perceptual network. *Computer* (1988).

[25] James Mercer. 1909. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* (1909).

[26] Yujie Mo, Liang Peng, Jie Xu, Xiaoshuang Shi, and Xiaofeng Zhu. 2022. Simple Unsupervised Graph Representation Learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

[27] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. 2016. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization. In *Neural Information Processing Systems (NeurIPS)*.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems (NeurIPS)*.

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

[30] Jeffrey Pennington, Felix Xinnan X Yu, and Sanjiv Kumar. 2015. Spherical Random Features for Polynomial Kernels. In *Neural Information Processing Systems (NeurIPS)*. https://proceedings.neurips.cc/paper_files/paper/2015/file/f7f580e11d00a75814d2ded41fe8e8fe-Paper.pdf

[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk. In *International Conference on Knowledge Discovery and Data Mining (KDD)*. https://doi.org/10.1145/2623330.2623732

[32] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. 2023. A critical look at the evaluation of GNNs under heterophily: Are we really making progress?. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=tJbbQfw-5wv

[33] Ali Rahimi and Benjamin Recht. 2007. Random Features for Large-Scale Kernel Machines. In *Neural Information Processing Systems (NeurIPS)*.

[34] Ali Rahimi and Benjamin Recht. 2008. Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning. In *Neural Information Processing Systems (NeurIPS)*. https://proceedings.neurips.cc/paper_files/paper/2008/file/0efe32849d230d7f53049ddc4a4b0c60-Paper.pdf

[35] David I Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* (2013).

[36] Nicolas Tremblay, Paulo Gonçalves, and Pierre Borgnat. 2017. Design of graph filters and filterbanks. *Cooperative and Graph Signal Processing* (2017).

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

[38] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*. http://arxiv.org/abs/2006.04131

A SUPPLEMENTARY MATERIAL

Contents

A.1 Reproducibility

We strive to ensure the reproducibility of our research findings. To facilitate this, we provide the details of our experimental setup, including dataset sources, preprocessing steps, hyperparameters, and model configurations. We also make our code and the datasets used, publicly available at this LINK, enabling researchers to reproduce our results and build upon our work. We would like to emphasize that our code is built on top of the existing MVGRL codebase. For the datasets used in our evaluation, we provide references to their original sources and any specific data splits that we employed. This allows others to obtain the same datasets and perform their own analyses using consistent data. Additionally, we specify the versions of libraries and frameworks used in our experiments, in Section A.3, and in the Requirements file and the Readme file, in the codebase, enabling others to set up a compatible environment. We document any specific seed values or randomization procedures that may affect the results. By providing these details and resources, we aim to promote transparency and reproducibility in scientific research. We encourage fellow researchers to reach out to us if they have any questions or need further clarification on our methods or results.

A.2 Datasets

Homophilic Datasets: We evaluated our model (as well as baselines) on three homophilic datasets: CORA, CITESEER, and PUBMED as borrowed from [17]. All three are citation networks, where each node represents a research paper and the links represent citations. Pubmed consists of medical research papers. The task is to predict the category of the research paper. We follow the same dataset setup mentioned in [29] to create 10 random splits for each of these datasets.

Heterophilic Datasets: In our evaluation, we included four heterophilic datasets: CHAMELEON, SQUIRREL, ROMAN-EMPIRE, and MINESWEEPER. For CHAMELEON and SQUIRREL, each node represents a Wikipedia web pages and edges capture mutual links between pages. We utilized the ten random splits provided in [29], where 48%, 32%, and 20% of the nodes were allocated for the train, validation, and test sets, respectively. In ROMAN-EMPIRE each node corresponds to a word in the Roman Empire Wikipedia article. Two words are connected with an edge if either these words follow each other in the text, or they are connected in the dependency tree of the sentence. The syntactic role of the word/node defines its class label. The MINESWEEPER graph is a regular 100x100 grid where each node is connected to eight neighboring nodes, and the features are one-hot encoded representations of the number of neighboring mines. The task is to predict which nodes are mines. For both ROMAN-EMPIRE and MINESWEEPER, we used the ten random splits provided in [32].

Large Datasets: We also evaluate our method on two large datasets OGBN-ARXIV (from [15]) and ARXIV-YEAR (from [22]). Both these datasets are from the arxiv citation network. In OGBN-ARXIV, the task is to predict the category of the research paper, and in ARXIV-YEAR the task is to predict the year of publishing. We use the publicly available splits for OGBN-ARXIV [17] and follow the same dataset setup mentioned in [22] to generate 5 random splits for ARXIV-YEAR. Note that OGBN-ARXIV is a homophilic dataset while ARXIV-YEAR is a heterophilic datasets.

The detailed dataset statistics can be found in Table 9.

A.3 Training Details

We conducted all experiments on a machine equipped with an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz processor, 440GB RAM, and a Tesla-P100 GPU with 16GB of memory. The experiments were executed using Python 3.9.12 and PyTorch 1.13.0 [28]. To optimize the hyperparameter search, we employed Optuna [1]. We utilized the Adam optimizer [19] for the optimization process.

A.3.1 Unsupervised Training. We conducted hyperparameter tuning for all unsupervised methods using 20 Optuna trials. The hyperparameter ranges and settings for each method are as follows:

DEEPWALK: We set the learning rate to 0.01, number of epochs to 20 and the varied the random walk length over $\{8, 9, 10, 11, 12\}$. Additionally, we varied the context window size over $\{3, 4, 5\}$ and the negative size (number of negative samples per positive sample) over $\{4, 5, 6\}$.

Table 9: Dataset Statistics. The table provides information on the following dataset characteristics: number of nodes, number of edges, feature dimension, number of classes, as well as the count of nodes used for training, validation, and testing.

	Heterophilic Datasets					Homophilic Datasets			
Properties	squirrel	chameleon	roman-empire	minesweeper	arXiv-Year	OGBN-arXiv	citeseer	pubmed	cora
#Nodes	5201	2277	22662	10000	169343	169343	3327	19717	2708
#Edges	222134	38328	32927	39402	1166243	1335586	12431	108365	13264
#Features	2089	500	300	7	128	128	3703	500	1433
#Classes	5	5	18	2	5	40	6	3	7
#Train	2496	1092	11331	5000	84671	90941	1596	9463	1192
#Val	1664	729	5665	2500	42335	29799	1065	6310	796
#Test	1041	456	5666	2500	42337	48603	666	3944	497

Node2Vec: For Node2Vec, we set the learning rate to 0.01 and number of epochs to 100. We varied the number of walks over $\{5, 10, 15\}$ and the walk length over $\{40, 50, 60\}$. The p (return parameter) value was chosen from $\{0.1, 0.25, 0.5, 1\}$ and q (in-out parameter) value was chosen from $\{3, 4, 5\}$.

DGI: DGI [37] proposes a self-supervised learning framework for graph representation learning by maximizing the mutual information between local and global structural context of nodes, enabling unsupervised feature extraction in graph neural networks. We relied on the authors' code[1] and the prescribed hyperparameter ranges specific to the DGI model, for our experiments.

MVGRL: MVGRL [11] proposes a method for learning unsupervised node representations by leveraging two views of the graph data, the graph diffusion view and adjacency graph view. We relied on the authors' code[2] and the prescribed hyperparameter ranges specific to the MVGRL model, for our experiments.

GRACE: GRACE [38] proposes a technique where two different perspectives of the graph are created through corruption, and the learning process involves maximizing the consistency between the node representations obtained from these two views. We relied on the authors' code[3] and the prescribed hyperparameter ranges specific to the GRACE model, for our experiments.

SUGRL: SUGRL [26] proposes a technique for learning unsupervised representations which capture node proximity, while also utilising node feature information. We relied on the authors' code[4] and the prescribed hyperparameter ranges specific to the SUGRL model, for our experiments.

FiGURe: We followed the setting of the MVGRL model, setting the batch size to 2 and number of GCN layers to 1. We further tuned the learning rate over $\{0.00001, 0.0001, 0.001, 0.01, 0.1\}$ and the sample size (number of nodes selected per batch) over $\{1500, 1750, 2000, 2250\}$, except for the large graphs, for which we set the sample size to 5000.

In each case, we selected the hyperparameters that resulted in the lowest unsupervised training loss.

A.3.2 Supervised Training. For all unsupervised methods, including the baselines and our method, we perform post-training supervised evaluation using logistic regression with 60 Optuna trials. We set the maximum number of epochs to 10000 and select the epoch and hyperparameters that yield the best validation accuracy. The learning rate is swept over the range $\{0.00001, 0.0001, 0.001, 0.0015, 0.01, 0.015, 0.1, 0.5, 1, 2\}$, and the weight decay is varied over $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0, 0.5, 1, 3\}$.

FiGURe: Along with the hyperparameters described above, following the approach described in [13], we also tune the combination coefficients (α_i's) with a separate learning rate. This separate learning rate is swept over the range $\{0.00001, 0.0001, 0.001, 0.0015, 0.01, 0.015, 0.1, 0.5, 1, 2\}$. In addition, we have a coefficient for masking the incoming embeddings from each filter, which is varied between 0 and 1. Furthermore, these coefficients are passed through an activation layer, and we have two options: 'none' and 'exp'. When 'none' is selected, the coefficients are used directly, while 'exp' indicates that they are passed through an exponential function before being used.

FiGURe with RFF: For the experiments involving Random Fourier Features (RFF), we use the same hyperparameter ranges as mentioned above. However, we also tune the gamma parameter which is specific to RFF projections. The gamma parameter is tuned within the range $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2\}$.

A.3.3 Negative Sampling for the Identity Filter. In our implementation of $\mathbf{F}_{\text{GPRGNN}}$ or $\mathbf{F}_{\text{BernNet}}$, we follow a specific procedure for handling the filters during training and evaluation. For all filters except the identity filter (**I**), we employ the negative sampling approach described in Section 6. However, the identity filter is treated differently. During training, we exclude the identity filter and only include it during evaluation.

During negative sampling, the generation of the negative anchor involves shuffling the node features, followed by premultiplying the shuffled node feature matrix with the filter matrix and computing the mean. On the other hand, for the positive anchor, the same procedure is applied without shuffling the node features. This approach encourages the model to learn meaningful patterns and relationships in the data when the filter matrix is not the identity matrix.

The decision to exclude the identity filter during training is based on the observation that it presents a special case where the positive and negative anchors become the same. As a result, the model would optimize and minimize the same quantity, potentially leading to trivial solutions. To prevent this, we exclude the identity filter during training.

[1] https://github.com/PetarV-/DGI.git
[2] https://github.com/kavehhassani/mvgrl.git
[3] https://github.com/CRIPAC-DIG/GRACE.git
[4] https://github.com/YujieMo/SUGRL.git

By excluding the identity filter during training, we ensure that the model focuses on the other filters in $\mathbf{F}_{\text{GPRGNN}}$ or $\mathbf{F}_{\text{BernNet}}$ to capture and leverage the diverse information present in the graph. Including the identity filter only during evaluation allows us to evaluate its contribution to the final performance of the model. This approach helps prevent the model from learning trivial solutions and ensures that it learns meaningful representations by leveraging the other filters.

A.4 Comparison with other Supervised Methods

Table 10 presents a comparison with common supervised baselines. Specifically, we choose 3 models for comparison, representing hr three different kinds of supervised methods, standard aggregation models (GCN), spectral filter-based models (GPRGNN) and smart-aggregation models (H_2GCN). There are two key observations from this table. Firstly, FiGURe is competitive with the supervised baselines, lagging behind only by a few percentage points in some cases. This suggests that much of the information that is required by the downstream tasks, captured by the supervised models, can be made available through unsupervised methods like FiGURe which uses filter banks. It is important to note that in FiGURe we only utilize logistic regression while evaluating on the downstream task. This is much more efficient that training a graph neural network end to end. Additionally it is possible that further gains may be obtained by utilizing a non-linear model like an MLP.

Furthermore, as indicated by 10, we can gain further computational efficiency by utilizing lower dimensional representations like 32 and 128 (with RFF), and still not compromise significantly on the performance.

Overall FiGURe manages to remain competitive despite not having access to task-specific labels and is computationally efficient as well.

A.5 RFF Projections

As shown in Section 6.2 and in Section 6.4, RFF projections are a computationally efficient way to achieve training by preserving the latent class behavior present in lower dimensional embeddings, by projecting them into a higher dimensional linearly separable space. The natural question that comes up is how do we compute these RFF projections? We provide an algorithm to compute the RFF projections in this section, in algorithm 1. Note that this follows [33].

Algorithm 1 Random Fourier Feature Computation

Require: Input data $X \in \mathbb{R}^{N\times d}$, target dimension D, kernel bandwidth γ

Ensure: Random Fourier Features $Z \in \mathbb{R}^{N\times D}$

1: Initialize random weight matrix $W \in \mathbb{R}^{d\times D}$ with Gaussian distribution
2: Initialize random bias vector $b \in \mathbb{R}^{D}$ uniformly from $[0, 2\pi]$
3: Compute scaled input $X' = \gamma XW + b$
4: Compute random Fourier features $Z = \sqrt{\frac{2}{D}}\cos(X')$
5: **return** Z

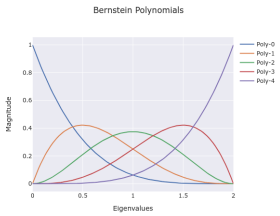

Figure 3: Five Bernstein Basis

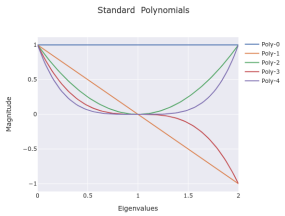

Figure 4: Five Standard Basis

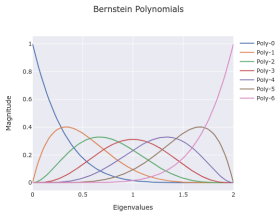

Figure 5: Seven Bernstein Basis

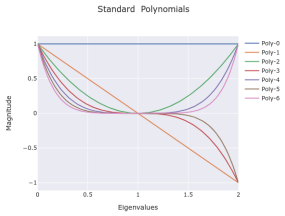

Figure 6: Seven Standard Basis

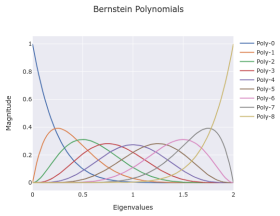

Figure 7: Nine Bernstein Basis

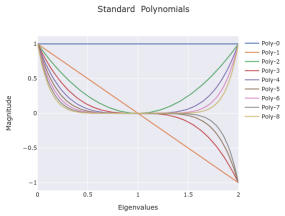

Figure 8: Nine Standard Basis

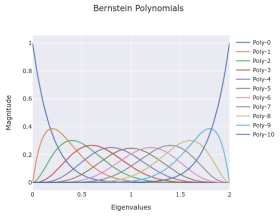

Figure 9: Eleven Bernstein Basis

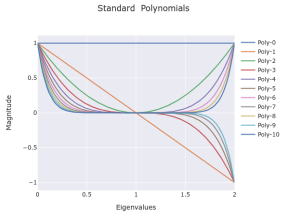

Figure 10: Eleven Standard Basis

Figure 11: The figures contain the Bernstein basis as well as standard basis for different degrees. The x-axis of the figures represents the eigenvalues of the Laplacian matrix, while the y-axis represents the magnitude of the polynomials. It is important to note that while plotting the standard polynomials, they are computed with respect to the Laplacian matrix (L_n) rather than the adjacency matrix. As a result, the eigenvalues lie between $[0, 2]$. On the other hand, the Bernstein polynomials are typically defined for the normalised Laplacian matrix, and therefore there is no change in the eigenvalue range (the eigenvalues of the normalised Laplacian matrix typically range from 0 to 2). By using the Laplacian matrix as the basis for plotting the polynomials, we can observe the behavior and magnitude of the polynomials at different eigenvalues, providing insights into their spectral properties and frequency response characteristics.

Table 10: Contains node classification accuracy percentages on heterophilic and homophilic datasets. GCN, GPRGNN and H_2GCN are supervised methods. F*i*GURe_{32} and F*i*GURe_{128} refer to F*i*GUR*e* trained with 32 and 128 dimensional representations, respectively, and then projected using RFF. The remaining models are trained at 512 dimensions. Higher numbers indicate better performance.

	Heterophilic Datasets					Homophilic Datasets			
	squirrel	chameleon	roman-empire	minesweeper	arXiv-Year	OGBN-arXiv	cora	citeseer	pubmed
GCN	47.78 (2.13)	62.83 (1.52)	73.69 (0.74)	89.75 (0.52)	46.02 (0.26)	69.37 (0.00)	87.36 (0.91)	76.47 (1.34)	88.41 (0.46)
GPRGNN	46.31 (2.46)	62.59 (2.04)	64.85 (0.27)	86.24 (0.61)	45.07 (0.21)	68.44 (0.00)	87.77 (1.31)	76.84 (1.69)	89.08 (0.39)
H_2GCN	37.90 (2.02)	58.40 (2.77)	60.11 (0.52)	89.71 (0.31)	49.09 (0.10)	OOM	87.81 (1.35)	77.07 (1.64)	89.59 (0.33)
F*i*GURe_{32}	48.89 (1.55)	65.66 (2.52)	67.67 (0.77)	85.28 (0.71)	41.30 (0.21)	66.58 (0.00)	82.56 (0.87)	71.25 (2.20)	84.18 (0.53)
F*i*GURe_{128}	48.78 (2.48)	66.03 (2.19)	68.10 (1.09)	85.16 (0.58)	41.94 (0.15)	69.11 (0.00)	86.14 (1.13)	73.34 (1.91)	85.41 (0.52)
F*i*GUR*e*	52.23 (1.19)	68.55 (1.87)	70.99(0.52)	85.58 (0.49)	42.26 (0.20)	69.69 (0.00)	87.00 (1.24)	74.77 (2.00)	88.60 (0.44)

A.6 Choice of Filter Banks

In Section 4.1, we explore the flexibility of FiGURe to accommodate various filter banks. When making a choice, it is crucial to examine the intrinsic properties of the filters contained within different filter banks. We pick two filter banks $\mathbf{F}_{\text{BernNet}}$ and $\mathbf{F}_{\text{GPRGNN}}$ and provide an overview of the filters contained in the filter banks. We use these two filter banks as examples to illustrate what should one be looking for, while choosing a filter bank.

Bernstein Polynomials: Figure 11 illustrates that as the number of Bernstein Basis increases, the focus on different parts of the eigenspectrum also undergoes changes. With an increase in polynomial order, two notable effects can be observed. Firstly, the number of filters increases, enabling each filter to focus on more fine-grained eigenvalues. This expanded set of polynomial filters allows for a more detailed examination of the eigenspectrum. Secondly, if we examine the first and last Bernstein polynomials, we observe an outward shift in their shape. This shift results in the enhancement of a specific fine-grained part at the ends of the spectrum. These observations demonstrate that Bernstein polynomials offer the capability to selectively target and enhance specific regions of interest within the eigenspectrum

Standard Basis: Figure 11 reveals two key observations. Firstly, at a polynomial order of 2, the standard basis exhibit focus at the ends of the spectrum, in contrast to the behavior of Bernstein polynomials, which tend to concentrate more on the middle of the eigenspectrum. This discrepancy highlights the distinct characteristics and emphasis of different polynomial bases in capturing different parts of the eigenspectrum. Secondly, as the number of polynomials increases (in contrast to Bernstein polynomials), the lower order polynomials remain relatively unchanged. Instead, additional polynomials are introduced, offering a more fine-grained focus at the ends of the spectrum. This expansion of polynomials allows for a more detailed exploration of specific regions of interest within the the ends of eigenspectrum.

In the context of filter banks, previous studies [6, 23] have demonstrated that certain datasets, such as squirrel and chameleon, benefit from frequency response functions that enhance the tail ends of the eigenspectrum. This observation suggests that the standard basis, which naturally focuses on the ends of the spectrum, may outperform Bernstein basis functions at lower orders. However, as the order of the Bernstein basis increases, as discussed in 4.1, there is a notable improvement in performance. This can be attributed to the increased focus of Bernstein basis functions on specific regions, particularly the ends of the spectrum. As a result, higher-order Bernstein filters exhibit enhanced capability in capturing important information in those regions. It is worth noting that the choice between $\mathbf{F}_{\text{GPRGNN}}$ and $\mathbf{F}_{\text{BernNet}}$ depends on the specific requirements of the downstream task. If the task necessitates a stronger focus on the middle of the spectrum or requires a band-pass or comb-like frequency response, $\mathbf{F}_{\text{BernNet}}$ is likely to outperform $\mathbf{F}_{\text{GPRGNN}}$. Thus, the selection of the appropriate filter bank should be based on the desired emphasis on different parts of the eigenspectrum. Regarding the performance comparison between $\mathbf{F}_{\text{BernNet}}$ and $\mathbf{F}_{\text{GPRGNN}}$, it is plausible that as we increase the order of the Bernstein basis, the performance could potentially match that of $\mathbf{F}_{\text{GPRGNN}}$. However, further investigation and experimentation are required to determine the specific conditions and orders at which this convergence in performance occurs.

A.7 Visualising RFF Behavior and Community Structure

As shown in prior sections, FiGURe improves on both computational efficiency as well as performance by utilising RFF projections. In this section, we aim to gain insights into the behavior of RFF projections and comprehend their underlying operations through a series of simple visualizations.

t-SNE Plots: Figure 15 offers insights into the structure of the embeddings for the cora dataset across different dimensions. Remarkably, even at lower dimensions (e.g., 32 dimensions), clear class structures are discernible, indicating that the embeddings capture meaningful information related to the class labels. Furthermore, when employing RFF to project the embeddings into higher dimensions, these distinct class structures are still preserved. This suggests that the role of RFF is not to introduce new information, but rather to enhance the suitability of lower-dimensional embeddings for linear classifiers while maintaining the underlying class-related information. Notably, even at 512 dimensions, the class structures remain distinguishable. However, it is worth noting that the class-specific embeddings appear to be more tightly clustered and less dispersed compared to the 32-dimensional embeddings or the projected 32-dimensional embeddings. This suggests that learning a 512-dimensional embedding differs inherently from learning

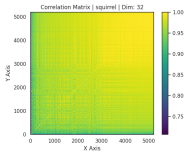

Figure 16: 32 Dims

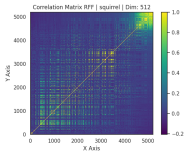

Figure 17: RFF

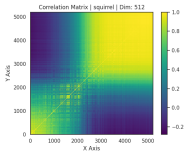

Figure 18: 512 Dims

Figure 19: The figures display the normalized correlation plots for the SQUIRREL dataset. These plots illustrate the normalized correlation values between embeddings generated by the F_3 filter. In the case of FiGURe, this filter corresponds to the square of the adjacency matrix (A^2). The normalized correlation provides a measure of similarity or agreement between the embeddings obtained using the F_3 filter for different embedding dimensions. These plots can help analyze the consistency or variation of embeddings across different dimensions and datasets. Note that Fig 16 illustrates the correlation plot of the 32 dimensional embeddings. Fig 17 illustrates the correlation plot of the 32 dimensional embeddings projected to 512 dimensions via RFF. Fig 18 illustrates the correlation plot of the 512 dimensional embeddings.

a 32-dimensional embedding and subsequently projecting it into higher dimensions.

Figure 12: 32 Dims

Figure 13: RFF

Figure 14: 512 Dims

Figure 15: The figures present t-SNE plots for the CORA dataset. These plots showcase the embeddings generated by the F_3 filter, which corresponds to A^2 in the case of FiGURe. The t-SNE plots are generated at different embedding dimensions, providing insights into the distribution and clustering of the embeddings for each dataset. Note that Fig 12 illustrates the t-SNE plot of the 32 dimensional embeddings. Fig 13 illustrates the t-SNE plot of the 32 dimensional embeddings projected to 512 dimensions via RFF. Fig 14 illustrates the t-SNE plot of the 512 dimensional embeddings.

Correlation Plots: Figure 19 offers insights into the correlation patterns within the embeddings generated from the SQUIRREL dataset across different dimensions. In lower dimensions, the embeddings exhibit high correlation with each other, which can be attributed to the presence of a mixture of topics or latent classes within the dataset. However, when the embeddings are projected to higher dimensions using RFF, the correlation is reduced, and a block diagonal matrix emerges. This block diagonal structure indicates the presence of distinct classes or communities within the dataset. Even at 512 dimensions, a more refined block diagonal structure can be observed compared to the correlation matrix of the 32-dimensional embeddings. Furthermore, it is noteworthy that the correlation of the projected embeddings can be regarded as a sparser version of the correlation observed in the 512-dimensional embeddings.