Automated Unsupervised Graph Representation Learning

Zhenyu Hou, Yukuo Cen, Yuxiao Dong, Jie Zhang, and Jie Tang*, *IEEE Fellow*

Abstract—Graph data mining has largely benefited from the recent developments of graph representation learning. Most attempts to improve graph representations have thus far focused on designing new network embedding or graph neural network (GNN) architectures. Inspired by the SGC and ProNE models, we instead focus on enhancing any existing or learned graph representations by further smoothing them via graph filters. In this paper, we introduce an automated framework AutoProNE to achieve this. Specifically, AutoProNE automatically searches for a unique optimal set of graph filters for any input dataset, and its existing representations are then smoothed via the selected filters. To make AutoProNE more general, we adopt self-supervised loss functions to guide the optimization of the automated search process. Extensive experiments on eight commonly used datasets demonstrate that the AutoProNE framework can consistently improve the expressive power of graph representations learned by existing network embedding and GNN methods by up to 44%. AutoProNE is also implemented in CogDL, an open source graph learning library, to help boost more algorithms.

Index Terms—Representation learning, Graph embedding, Graph filter

1 Introduction

GRAPHS have been commonly used for modeling structured and relational data, such as social networks, knowledge graphs, and biological networks. Over the past few years, mining and learning with graph data have been shifted from structural feature engineering to graph representation learning, which offers promising results in various applications, including node classification [1], recommender system [2], and machine cognitive reasoning [3].

Broadly, research on graph representation learning can be grouped into two categories: unsupervised network embedding methods and graph neural networks (GNNs). The network embedding methods aim to project the structure of a network into a latent low-dimensional space such as its structural properties are encoded in the latent vectors. Usually, the input to them contains only the graph topology without input features. Representative network embedding models include skip-gram based methods such as DeepWalk [4], LINE [5], and node2vec [6] as well as matrix factorization based methods such as HOPE [7], and NetMF [8].

Graph neural networks (GNNs) on the other hand usually take both node/edge features and graph structures as the input and iteratively aggregate neighbors' features to update each node's representation. Most early GNN models are end-to-end (semi) supervised frameworks with the label information for optimization, such as the graph convolutional network (GCN) [1] and graph attention network (GAT) [9]. Recently, self-supervised graph neural networks gain significant attention due to their enormous potential in shaping the future of graph mining and learning. For example, the GraphSage model [10] with the unsupervised loss can be considered as a self-supervised framework. DGI [11] and GCC [12] leverage the idea of contrastive learning [13], [14] to pre-train graph neural networks via self-supervised signals from the unlabeled input data.

Among the exciting progress in graph representation learning, two recent developments are particularly attractive. First, Wu et al. [15] discover that the non-linearity between GCN layers can be simplified, based on which the SGC model without non-linearity is proposed. By removing the non-linearity, it is easy to decouple the feature transformation and propagation steps in GCNs' neighborhood aggregation process, enabling us to design these two steps separately. The second one is the ProNE model, in which Zhang et al. [16] show that using a modulated Gaussian filter to propagate/smooth the node embeddings can significantly improve the expressive power of the embeddings.

Inspired by these two works, we propose to study whether we can improve graph representation learning by focusing on the propagation step, that is, given the input embeddings such as learned by DeepWalk or DGI, the goal is to further enhance their expressive power by propagating/smoothing the embeddings. First, instead of relying on label information, we are interested in techniques that can benefit graph representation learning in an unsupervised or self-supervised manner. Second, rather than a fixed Gaussian filter, we would like to answer the question of whether there exist better graph filters for propagating existing embeddings. Finally, the natural need is to avoid the manual design or choice of graph filters for each dataset, that is, can we automatically find the best filters for each specific graph?

Solutions. To address these issues, we present the AutoProNE framework to automatically find the appropriate graph filters to smooth existing graph representations in a self-supervised manner. Different from ProNE that uses a fixed Gaussian kernel for all graphs, the proposed framework leverages the idea of AutoML

- *Zhenyu Hou, Yukuo Cen and Jie Zhang are with the Department of Computer Science and Technology, Tsinghua University, China. E-mail:{houzy21, cyk20, j-z16}@mails.tsinghua.edu.cn.*
- *This work was done when Yuxiao Dong was at Microsoft Research, Redmond, and he is now at Facebook AI, USA. Email: ericdongyx@gmail.com*
- *Jie Tang is with the Department of Computer Science and Technology, Tsinghua University, and Tsinghua National Laboratory for Information Science and Technology (TNList), China. E-mail: jietang@tsinghua.edu.cn, corresponding author.*

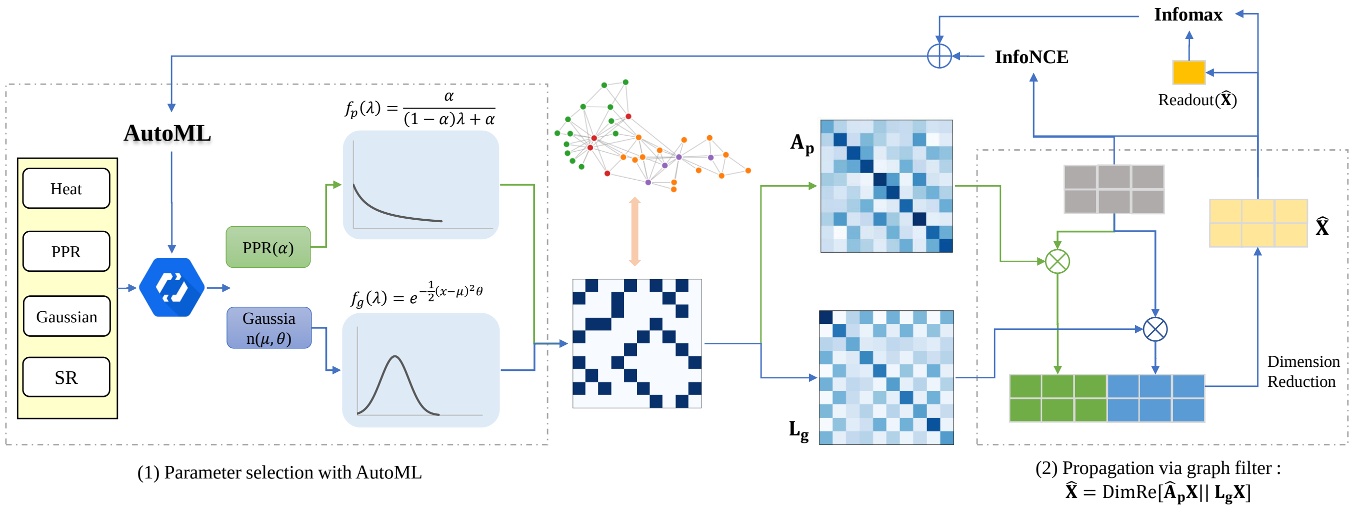

Infomax
InfoNCE
Readout($\hat{\mathbf{X}}$)
AutoML
Heat
PPR
Gaussian
SR
$f_p(\lambda) = \frac{\alpha}{(1-\alpha)\lambda + \alpha}$
PPR(α)
Gaussia n(μ, θ)
$f_g(\lambda) = e^{-\frac{1}{2}(x-\mu)^2\theta}$
$\mathbf{A_p}$
$\mathbf{L_g}$
$\hat{\mathbf{X}}$
Dimension Reduction
(1) Parameter selection with AutoML
(2) Propagation via graph filter : $\hat{\mathbf{X}}$ = DimRe[$\hat{\mathbf{A}}_\mathbf{p}\mathbf{X}$|| $\mathbf{L_g X}$]

Fig. 1. The Overview of the AutoProNE Framework. The input is the node embeddings learned by any other network embedding or GNN methods and the output is the smoothed embeddings with enhanced expressive power. AutoProNE includes two components: (1) Parameter selection: the AutoML parameter generator automatically select graph filters and corresponding hyperparameters. (2) Embedding propagation: once the graph filters are selected, they are used to smooth the input embeddings. In the example, the PPR and Gaussian graph filters as well as their parameters are selected for the specific input graph. And as a result, each node pays attention to its indirect neighbors.

to search for the best filter(s) for the given graph, such as low-pass, band-pass, and signal-rescaling filters. Moreover, rather than using supervised signals to guide the AutoML process, the optimization of AutoProNE is built upon the recent advancements in self-supervised graph representation learning, that is, AutoProNE adopts self-supervised loss to direct the AutoML optimization. Figure 1 illustrates the framework of AutoProNE.

We conduct extensive experiments on eight publicly available datasets. By using both network embedding methods without input features (e.g., DeepWalk) and graph neural networks with node features (e.g., DGI), we show that the AutoProNE framework can consistently and significantly enhance the expressive power of graph representations learned by these base methods. For example, the simple, automated, and unsupervised AutoProNE can boost the node classification performance of LINE's representations by 34–44% on the PPI dataset. Furthermore, we show that the graph filters automatically decided by the AutoML process of AutoProNE are always among the best options across all different datasets. Finally, we find that AutoProNE requires only 2–4% of the running time of DeepWalk to offer outperformance by up to 8%, and also demonstrate that its scalability is linear to the number of nodes in the graph, enabling it for handling large-scale data. The code is available at https://github.com/THINK2TRY/AutoProNE. and CogDL [1] [17] , a open source graph learning library, to make it more convenient to collaborate with existing graph representation algorithms.

Contribution. The main contributions of this work include:

- We investigate the role of graph filters on unsupervised graph representation learning and provide insights into various graph filters.
- We propose a comprehensive framework AutoProNE that integrates automatic machine learning and graph representation learning.
- We conduct extensive experiments to evaluate our framework. The results demonstrate the effectiveness of AutoProNE where our framework achieves significant improvements over various methods and datasets.

1. https://github.com/THUDM/cogdl

2 Preliminaries

2.1 Concepts

Graph Notations. Let $\mathcal{G}$=$(\mathcal{V}, \mathcal{E}, \mathbf{X})$ denote an undirected graph with $\mathcal{V}$ as the set of its n nodes and $\mathcal{E}$ as its edges, and $\mathbf{X} \in \mathbb{R}^{n\times d}$ as the feature matrix of $\mathcal{V}$. Given $\mathcal{G}$, we have its (symmetric) adjacency matrix as $\mathbf{A} = (a_{ij})$ with a_{ij} = 1 if and only if there exists an edge between v_i and v_j, as well as its degree matrix $\mathbf{D} = \text{diag}(d_1, d_2, ..., d_n)$ with $d_i = \sum_j a_{ij}$. In practice, the row-normalized adjacency matrix $\hat{\mathbf{A}}_{rw} = \mathbf{D}^{-1}\mathbf{A}$ and symmetric normalized adjacency matrix $\hat{\mathbf{A}}_{sym} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ are more commonly used. In this work, we simplify $\hat{\mathbf{A}}_{rw}$ and $\hat{\mathbf{A}}_{sym}$ as $\hat{\mathbf{A}}$.

Network Embedding. The problem of network embedding aims to learn a mapping function $f : \mathcal{V} \rightarrow \mathbb{R}^d$ that projects each node to a d-dimensional space ($d \ll |\mathcal{V}|$). Network embedding methods mainly focus on neighborhood similarity and capture the structural properties of the network. Extensive studies have shown that the learned node representations can benefit various graph mining tasks, such as node classification and link prediction. DeepWalk, LINE, Node2Vec and NetMF are all network embedding methods.

Graph Signal Processing. In this part, we introduce a recent formulation [18] of graph signal processing [19], [20] on irregular graphs. The Laplacian matrix of graph $\mathcal{G}$ is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. $\hat{\mathbf{L}} = \mathbf{I}_n - \hat{\mathbf{A}}$ is the augmented normalized Laplacian. A vector $\boldsymbol{x} \in \mathbb{R}^n$ defined on the nodes of $\mathcal{G}$ is called a *graph signal*. A useful property of graph Laplacian $\mathbf{L}$ is that its quadratic form measures the *smoothness* of the graph signal. It is easy to verify that

$$\boldsymbol{x}^T\mathbf{L}\boldsymbol{x} = \sum_{i,j} \mathbf{A}_{ij}(x_i - x_j)^2 = \sum_{(i,j)\in\mathcal{E}} (x_i - x_j)^2 \qquad (1)$$

Let $\mathbf{U} \in \mathbb{R}^{n\times n} = [\boldsymbol{u}_1, ..., \boldsymbol{u}_n]^T$ be the eigenvectors of $\hat{\mathbf{L}}$ and we have $\hat{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, where $\mathbf{\Lambda} = \text{diag}[\lambda_1, ..., \lambda_n]$ is the eigenvalues of $\hat{\mathbf{L}}$ corresponding to $\mathbf{U}$. The *graph Fourier transform* $\mathcal{F}: \mathbb{R}^n \to \mathbb{R}^n$ is defined by $\mathcal{F}\boldsymbol{x} = \tilde{\boldsymbol{x}} = \mathbf{U}^T\boldsymbol{x}$, and the *inverse graph Fourier transform* $\mathcal{F}^{-1}$ is $\mathcal{F}^{-1}\tilde{\boldsymbol{x}} = \boldsymbol{x} = \mathbf{U}\tilde{\boldsymbol{x}}$ because $\mathcal{F}\mathcal{F}^{-1} = \mathbf{U}^T\mathbf{U} = \mathbf{I}_n$.

Graph filter is defined based on the graph Fourier transform. Let g: $\mathbb{R} \to \mathbb{R}$ be a function mapping $x \xrightarrow{g} y$. In frequency domain, the graph filter specified by g is defined by the relation: $\tilde{\boldsymbol{y}} = g(\boldsymbol{\Lambda})\tilde{\boldsymbol{x}}$, that is, $\tilde{\boldsymbol{y}}(\lambda_i) = g(\lambda_i)\tilde{\boldsymbol{x}}(\lambda_i)$. In the spatial domain, the above equation is equivalent to $\boldsymbol{y} = g(\hat{\mathbf{L}})\boldsymbol{x}$.

For multi-dimensional cases, the graph Fourier transform is $\mathcal{F}\mathbf{X} = \tilde{\mathbf{X}} = \mathbf{U}^T\mathbf{X}$ and the inverse form is $\mathcal{F}^{-1}\tilde{\mathbf{X}} = \mathbf{X} = \mathbf{U}\tilde{\mathbf{X}}$. Then the graph filter is represented as

$$\mathbf{Y} = g(\hat{\mathbf{L}})\mathbf{X} = \mathbf{U}g(\boldsymbol{\Lambda})\mathbf{U}^T\mathbf{X} \tag{2}$$

Generally, for computational efficiency, Taylor or Chebyshev approximation is applied to avoid the eigendecomposition of $\hat{\mathbf{L}}$. Without loss of generality, let $\mathbf{T} \in \mathbb{R}^{n\times n}$ be the transition matrix and θ_i be the weight coefficients. The k-order expansion is represented as

$$g(\hat{\mathbf{L}}) \approx \sum_{i=0}^{k} \theta_i \mathbf{T}^i \in \mathbb{R}^{n\times n} \tag{3}$$

Graph Convolution. In spectral graph convolution networks, the graph convolution operation is fast approximated with the layer-wise propagation rule [1]. GCN simplifies Eq 3 by only keeping the first two items with $\theta_0 = \theta_1$:

$$\begin{aligned} g(\hat{L}) &= \theta_0\mathbf{I_n} + \theta_1\hat{\mathbf{A}} \\ &= \theta(\mathbf{I_n} + \hat{\mathbf{A}}) \end{aligned} \tag{4}$$

$\mathbf{I}_n$ is an identity matrix. The eigenvalues of $\mathbf{I} + \hat{\mathbf{A}}$ is in the range [0, 2]. To alleviate the problem of numerical instability and exploding/vanishing gradients when used in deep neural networks, the following *renormalization trick* is introduced:

$$\mathbf{I_n} + \hat{\mathbf{A}} \to \tilde{\mathbf{D}}^{-1/2}(\mathbf{I} + \mathbf{A})\tilde{\mathbf{D}}^{-1/2} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$$

where $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. Furthermore, multiple layers are stacked to recover a rich class of convolutional filter functions and each layer is followed by a linear transformation and a point-wise non-linearity. Then the one-layer graph convolution becomes as follows:

$$\mathbf{H}^{(i+1)} = \sigma(\tilde{\mathbf{A}}\mathbf{H}^{(i)}\boldsymbol{\Theta}^{(i)}) \tag{5}$$

where $\boldsymbol{\Theta}^{(i)}$ is a layer-specific trainable matrix, $\mathbf{H}^{(i)} \in \mathbb{R}^{n\times d}$ is the d-dimensional hidden node representation in the i^{th} layer.

2.2 ProNE

In this part, we give a brief introduction to ProNE. ProNE is a fast network embedding method based on matrix factorization. First, it formulates network embedding as a sparse matrix factorization for efficient representation transformation. Second, it utilizes a Gaussian graph filter for representation smoothing to improve the representation.

Network Embedding as Sparse Matrix Factorization ProNE leverages an edge to represent a node-context pair. Let $r_i, c_i \in \mathbb{R}^d$ be the node embedding and context vectors of node v_i respectively. The concurrence of context v_j given v_i is

$$\hat{p}_{i,j} = \sigma(r_i^T c_j) \tag{6}$$

$\sigma()$ is the sigmoid function. To maximize the concurrence probability and avoid trivial solution ($r_i = c_i \& \hat{p}_{i,j} = 1$) for each pair, the objective can be expressed as the weighted sum of log loss over all edges accompanied with negative sampling

$$Loss = -\sum_{(i,j)\in\mathcal{E}} [p_{i,j} \ln\sigma(r_i^T c_j) + \tau P_{\mathcal{E},j} \ln\sigma(\sigma(-r_i^T c_j))] \tag{7}$$

where $p_{i,j} = \mathbf{A}_{i,j}/\mathbf{D}_{i,i}$ indicates the weight of pair (v_i, v_j), τ is the ratio negative sampling, $P_{\mathcal{E},j} \propto (\sum_{i:(i,j)\in\mathcal{E}} p(i,j))^\alpha$ with $\alpha = 1$ or 0.75. To minimize the objective equals to let the the partial derivative with respect to $r_i^T c_j$ be zero. And hence we get

$$r_i^T c_j = \ln p_{i,j} - \ln(\tau P_{\mathcal{E},j}) \tag{8}$$

ProNE proposes to define a proximity matrix $\mathbf{M}$ with each entry as $r_i^T c_j$, which represents the similarity between embedding of v_i and context embedding of v_j.

$$\mathbf{M} = \begin{cases} \ln p_{i,j} - \ln(\tau P_{\mathcal{E},j}), & (v_i, v_j) \in \mathcal{E} \\ 0, & (v_i, v_j) \notin \mathcal{E} \end{cases}$$

Now that the relationship matrix is built, the objective is transformed into matrix factorization. ProNE uses a fast randomized tSVD to achieve fast sparse matrix factorization and finally gets $\hat{\mathbf{X}} \in \mathbb{R}^{N\times d}$ as node embedding matrix.

Spectral Propagation To address the representation smoothing problem, ProNE proposes to leverage a Gaussian graph filter as the smoothing function $f(\hat{\mathbf{A}})$. ProNE designs the graph filter as $g(\lambda) = e^{-[\frac{1}{2}(\lambda-\mu)^2-1]\theta}$ and formulates the transformation as the following rule:

$$f(\hat{\mathbf{A}}) = \mathbf{D}^{-1}\hat{\mathbf{A}}(\mathbf{I}_n - \tilde{\mathbf{L}}) \tag{9}$$

where $\mathbf{I}_n$ is the identity matrix and $\tilde{\mathbf{L}}$ is defined as the Laplacian filter as:

$$\tilde{\mathbf{L}} = \mathbf{U}\mathrm{diag}([g(\lambda_1), ..., g(\lambda_n)])\mathbf{U}^T \tag{10}$$

Eq 9 can be viewed as two steps. It first modulates the spectral property with modulator $\mathbf{I}_n - \tilde{\mathbf{L}}$ and then re-propagates information between neighbors.

3 Automated Unsupervised Graph Representation Learning

3.1 Problem

One GCN layer consists of three steps: neighborhood feature aggregation, feature transformation,and nonlinear transition. By stacking multiple layers, GCNs enable the propagation to reach high-order neighbors. Recently, Wu et al. [15] suggest that the non-linearity between GCN layers is not critical and thus propose the simplified graph convolution network (SGC) by removing non-linearity and having only one step of feature propagation and transformation, i.e.,

$$\mathbf{H}_{SGC} = \hat{\mathbf{A}}^K\mathbf{X}\boldsymbol{\Theta}_1 \cdots \boldsymbol{\Theta}_K = \hat{\mathbf{A}}^K\mathbf{X}\boldsymbol{\Theta} \tag{11}$$

Inspired by SGC, we can further abstract graph convolution as:

$$\mathbf{H} = \hat{\mathbf{A}}\mathbf{X}\boldsymbol{\Theta} = f(\hat{\mathbf{A}})h(\mathbf{X}, \boldsymbol{\Theta}) \tag{12}$$

where $f(\hat{\mathbf{A}})$ and $h(\mathbf{X}, \boldsymbol{\Theta})$ are two independent steps for feature representation learning. To make Eq. (12) generalize to unsupervised graph embedding methods, such as DeepWalk and node2vec, we have $h(\cdot)$ as $h(\mathbf{A}, \mathbf{X}, \boldsymbol{\Theta})$. In these methods, the input feature matrix $\mathbf{X}$ is empty and the node representations are learned only based on the graph topology $\mathbf{A}$. Therefore we can have

$$\mathbf{H} = f(\hat{\mathbf{A}})h(\mathbf{A}, \mathbf{X}, \boldsymbol{\Theta}) \tag{13}$$

In other words, graph representation learning, including both GNNs and unsupervised methods, can be simplified and abstracted as two independent processes: representation transformation $h(\mathbf{A}, \mathbf{X}, \boldsymbol{\Theta})$ and propagation (or smoothing) $f(\hat{\mathbf{A}})$.

In *representation transformation* $\hat{\mathbf{X}} = h(\mathbf{A}, \mathbf{X}, \boldsymbol{\Theta})$, the node representations $\hat{\mathbf{X}}$ are either learned solely from graph structures without input features $\mathbf{X}$ (such as by DeepWalk or NetMF) or transformed with $\mathbf{X}$ by learnable parameters (such as MLP or GNNs).

In the *representation smoothing* $f(\hat{\mathbf{A}})$ step, the representations initialized in the previous step are propagated to (high-order) neighbors. Note that in traditional unsupervised methods, this step is ignored. In SGC, the features are smoothed via the high order propagation, that is, $\hat{\mathbf{A}}^K$.

Problem. The focus of this work is on the smoothing step of graph representation learning. Specifically, given the node representations of one graph dataset learned by existing methods, such as DeepWalk, the goal is *to automatically design a smoothing function* $f(\hat{\mathbf{A}})$ *for this graph to effectively propagate them over its specific structure in an unsupervised manner.*

3.2 The AutoProNE Framework

Inspired by ProNE, we propose that graph signal processing can be a general and powerful method to address the representation smoothing problem. Graph filters, such as the low-pass filter and band-pass filter in the frequency domain, or adjusting the structure importance of nodes in the spatial domain can be used to design the smoothing function $f(\hat{\mathbf{A}})$.

ProNE uses a fixed Gaussian filter to tackle all cases. However, intuitively, each dataset may require unique graph filters to help “pass” true features and “filter out” noises. However, the input dataset is often considered as a black box and thus it is computationally expensive to analyze its spatial and spectral features. Therefore, it is infeasible to manually select or design appropriate graph filters for each given graph dataset.

In light of these challenges, we propose the automated AutoProNE graph representation learning framework to automatically select graph filters in an unsupervised manner for different input graphs. Fig 1 shows the overall architecture of AutoProNE, which first utilizes the idea of AutoML to select suitable graph filters for modulating graph signals (parameter selection) and then smooth node representations based on the selected filters (propagation).

In *parameter selection*, the parameter controller automatically selects graph filters from the designated graph filter set together with their hyper-parameters (Cf next section for details). We employ AutoML to implement this process. Briefly, AutoML is designed to automatically build machine learning applications [21]. It involves two parts—model generation and model evaluation.

First, model generation can be divided into search space and optimization methods, which are usually classified into hyperparameter optimization (HPO) and architecture optimization (AO). In AutoProNE, we mainly focus on AO, which indicates the model-related parameters, e.g., the selection of graph filters.

Second, for model evaluation, indicators like accuracy on the validation set are often used as measures. However, our problem is formalized under the unsupervised setting, that is, there exist no supervised indicators to help select the model parameters. To address this issue, we leverage the idea of contrastive learning to maximize the mutual information of different embeddings to evaluate the model.

In *propagation*, the node representations $\hat{\mathbf{X}}$ are propagated with selected filters. Under the hypothesis that each graph filter would differently amplify true features and attenuate noises, we concatenate the results of each filter together to preserve the information instead of averaging them, if multiple filters are selected. In order to keep the embedding dimension the same as the input, it is then followed by the SVD operation on the concatenated representations.

Finally, the loss is collected to help optimize the search. In AutoProNE, we directly utilize Optuna [22] as the AutoML framework. As the designed search space is small, the algorithm will converge quickly. Algorithm 1 illustrates the process of AutoProNE described above.

Note that AutoNE [23] also combines network embedding (NE) with AutoML and attempts to decide the hyperparameters of a given NE algorithm using meta-learning. Differently, AutoProNE aims to obtain the optimal graph filters for further improving the embeddings learned by any existing NE algorithms, instead of searching for hyperparameters to have better NE algorithms. In other words, AutoProNE runs on the embedding results and is not related to the learning process of the NE algorithms.

Algorithm 1 The AutoProNE Algorithm

```
Input: Normalized adjacency matrix $\hat{\mathbf{A}}$; Input embeddings $\hat{\mathbf{X}}$;
    Search iterations $N$.
Output: Enhanced embeddings $\mathbf{H}$ with the same dimension size
    as $\hat{\mathbf{X}}$.
 1: for $i = 1$ to $N$ do
 2:     Select filters $GFs$ from $\{PPR, Heat, Gaussian, SR\}$
 3:     Let $\mathbf{Z}$ be an empty list: $\mathbf{Z} = []$
 4:     for each $gf \in GFs$ do
 5:         $\mathbf{H}_{gf} = Propagation(\hat{\mathbf{A}}, \hat{\mathbf{X}}, gf)$
 6:         Append $\mathbf{H}_{gf}$ into $\mathbf{Z}$
 7:     end for
 8:     Concatenate smoothed embeddings in $\mathbf{Z}$ to have $\mathbf{Z_r}$
 9:     Conduct dimension reduction on $\mathbf{Z_r}$ to get $\mathbf{H}$
10:     Calculate the corresponding loss and optimize parameters.
11: end for
```

3.3 Automated Graph Filter Selection

We introduce the design choices of the core components in AutoProNE: *Search Space Design* and *Unsupervised Loss Function*.

3.3.1 Search Space Design.

In graph signal processing, different graph filters have been designed for modulating graph signals. We adopt three existing simple and efficient graph filters—*PPR, Heat kernel*, and *Gaussian kernel*—that have been widely used in graph representation learning [16], [24], [25] and also propose a new filter based on signal rescaling. The four graph filters are summarized in Table 1.

PPR [26]. Personalized PageRank(PPR) is defined based on random walk with restart and reflects the probability of a random walk starting from a node to another. PPR is defined as $\pi(\boldsymbol{x}) = (1-\alpha)\hat{\mathbf{A}}\pi(\boldsymbol{x}) + \alpha\boldsymbol{x}$, and its closed-form matrix is $\mathbf{A}_p = \alpha(\mathbf{I}_n - (1-\alpha)\hat{\mathbf{A}})$. To avoid the high computational cost

TABLE 1
Graph Filters

Name	Graph Filter	Parameters
PPR	$\mathbf{H} = \mathbf{A}_p\hat{\mathbf{X}}$	α
Heat Kernel	$\mathbf{H} = \mathbf{L}_h\hat{\mathbf{X}}$	θ
Gaussian Kernel	$\mathbf{H} = \mathbf{L}_g\hat{\mathbf{X}}$	μ, θ
Signal Rescaling	$\mathbf{H} = \mathbf{A}_s\hat{\mathbf{X}}$	-

of matrix inversion, we use Taylor expansion to approximate $\mathbf{A}_p$, that is,

$$\mathbf{A}_p \approx \sum_{i=0}^{k} \theta_p^{(i)} \hat{\mathbf{A}}, \quad where\ \theta_p^{(i)} = \alpha(1-\alpha)^i \tag{14}$$

Heat Kernel [27]. The heat kernel is often used in heat condition and diffusion, and represents the evolution of temperature in a region whose boundary is held fixed at a particular temperature. When applied to graphs, Heat kernel is defined as $f_h(\lambda) = e^{-\theta\lambda}$, in which θ is a scaling hyperparameter. We denote $f_h(\mathbf{\Lambda})$ as $f_h(\mathbf{\Lambda}) = \mathrm{diag}(e^{-\theta\lambda_1}, \dots, e^{-\theta\lambda_n})$. In fact, heat kernel works as a low-pass filter in the frequency domain and the heat graph filter is defined based on the Laplacian matrix as

$$\mathbf{L}_h = \mathbf{U} f_h(\mathbf{\Lambda})\mathbf{U}^T = \mathbf{U}\mathrm{diag}(e^{-\theta\lambda_1}, ..., e^{-\theta\lambda_n})\mathbf{U}^T \tag{15}$$

Gaussian Kernel [28]. Gaussian filter is a widely used band-pass filter in signal processing. Gaussian kernel in graph is defined as $f(\lambda) = e^{-[\frac{1}{2}(\lambda-\mu)^2-1]\theta}$ with μ and θ as the scaling hyperparameters. Gaussian kernel works as a band-pass filter with different hyperparameters. We have $f_g(\mathbf{\Lambda}) = \mathrm{diag}(e^{-\theta\bar{\lambda}_i}, \dots, e^{-\theta\bar{\lambda}_i})$ if we denote $\bar{\lambda}_i = \frac{1}{2}(\lambda_i - \mu)^2 - 1$. Then the Gaussian graph filter is similar to the heat graph filter:

$$\mathbf{L}_g = \mathbf{U} f_g(\mathbf{\Lambda})\mathbf{U}^T = \mathbf{U}\mathrm{diag}(e^{-\theta\bar{\lambda}_1}, ..., e^{-\theta\bar{\lambda}_n})\mathbf{U}^T \tag{16}$$

Signal Rescaling. In addition to the three existing filters, we also propose a signal rescaling filter. The intuition is that structural information plays a central role in networks. An intuitive example is that nodes of a high degree may be more important and influential in a network. To capture this phenomenon, we can attenuate node signals (and perform renormalization) before propagation, that is,

$$\mathbf{A}_s = \mathrm{Normalize}(\hat{\mathbf{A}}\sigma(\mathbf{D}^{-1})) \tag{17}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$.

Chebyshev Expansion for Efficiency For both the heat kernel and Gaussian kernel, we utilize the Chebyshev expansion and Bessel function [29] to avoid explicit eigendecomposition. Chebyshev polynomials of the first kind are defined as $T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x)$ with $T_0(x) = 1, T_1(x) = x$. Then, the expression of a general graph filter can be expanded as:

$$\bar{\mathbf{L}} \approx \mathbf{U}\sum_{i=0}^{k} c_i(\theta) T_i(\tilde{\mathbf{\Lambda}})\mathbf{U}^T = \sum_{i=0}^{k} c_i(\theta) T_i(\tilde{\mathbf{L}}) \tag{18}$$

where specifically $\tilde{\mathbf{\Lambda}} = \mathbf{\Lambda}$, $\tilde{\mathbf{L}} = \hat{\mathbf{L}}$ for heat kernel and $\tilde{\mathbf{\Lambda}} = \frac{1}{2}(\mathbf{\Lambda}-\mu\mathbf{I})^2$, $\tilde{\mathbf{L}} = \frac{1}{2}(\hat{\mathbf{L}}-\mu\mathbf{I})^2$ for Gaussian kernel. The coefficient $c_i(\theta)$ can be obtained with the Bessel function:

$$c_i(\theta) = \frac{\beta}{\pi}\int \frac{T_i(x)e^{-x\theta}}{\sqrt{1-x^2}}dx = \beta(-)^i B_i(\theta) \tag{19}$$

where β=1 if i=0 otherwise β=2 and $B_i(\theta)$ is the modified Bessel function of the first kind [29]. Then the truncated expansion of the filter becomes as follows:

$$\bar{\mathbf{L}} \approx B_0(\theta)T_0(\tilde{\mathbf{L}}) + 2\sum_{i=1}^{k-1} B_i(\theta)T_i(\tilde{\mathbf{L}}) \tag{20}$$

This truncated Chebyshev expansion provides a good approximation for $e^{-\lambda\theta}$ with a fast convergence rate.

Taylor expansion and Chebyshev expansion are common approximation methods. [30] adopts another local spectral embedding method which utilizes random vectors and Gauss-Seidel iteration to avoid explicit eigendecomposition of the adjacency matrix.

3.3.2 *Unsupervised Loss Function.*

Different from most AutoML settings where supervised information is used, we aim to guide the AutoML process for graph filter selection in an unsupervised or self-supervised fashion. The goal is to derive supervised signals from the input graph data itself for "supervising" the selection of graph filters.

An additional principle is that the self-supervised loss function Q_f of AutoML in the smoothing phase should be different from the one utilized in getting the representations in the transformation phase. The reason lies in that the representations to be smoothed have already achieved the best loss in the transformation step, and a different Q_f can help further improve the node representations.

In AutoProNE, our strategy for automatic graph filter search is built upon the recent development of self-supervised learning techniques, specifically contrastive learning based methods [11], [12], [31]. Note that AutoProNE is generally designed for improving node representations learned by both network embedding methods (e.g., DeepWalk) and GNNs. To accommodate their unique properties, we employ the mutual information maximization and instance discrimination as the loss function for both sets of methods, respectively.

For unsupervised network embedding methods, such as skip-gram or matrix factorization based methods, we use the local-global mutual information maximization loss proposed in [11] to guide the search of graph filters for each graph. Let $\tilde{\mathbf{Z}}$ be the row-wise shuffling of $\hat{\mathbf{X}}$. $\hat{\mathbf{X}}$ and $\tilde{\mathbf{Z}}$ are both propagated with selected graph filters, with the results as $\mathbf{H} = Prop(\hat{\mathbf{X}})$ and $\tilde{\mathbf{H}} = Prop(\hat{\mathbf{Z}})$ respectively. As $\tilde{\mathbf{H}}$ is derived from the propagation based on shuffled features, it is "corrupted" and should be less similar to initial embeddings $\hat{\mathbf{X}}$. We leverage the mean-readout function to obtain a graph level representation: $\boldsymbol{s} = \frac{1}{N}\sum_{\boldsymbol{h}\in\mathbf{H}} \boldsymbol{h}$. In such case, $\mathbf{H}$ is considered as "positive" samples for $\boldsymbol{s}$, $\tilde{\mathbf{H}}$ as "negative" samples for $\tilde{\mathbf{H}}$ is derived from shuffled features. The goal is to maximize the mutual information between node features $\mathbf{H}$ and global features $\boldsymbol{s}$ in the mean time minimizing the mutual information between $\boldsymbol{s}$ and negative node features $\tilde{\mathbf{H}}$. Then the loss is formalized as follows:

$$\begin{aligned} Q_f = -\frac{1}{2N}(&\sum_{i=1}^{N}\mathbb{E}_{(\hat{\mathbf{X}},\hat{\mathbf{A}})}[\log\sigma(\boldsymbol{h}_i^T\boldsymbol{s})] \\ &+ \sum_{j=1}^{N}\mathbb{E}_{(\tilde{\mathbf{Z}},\hat{\mathbf{A}})}[\log\sigma(1-\tilde{\boldsymbol{h}}_j^T\boldsymbol{s})]) \end{aligned} \tag{21}$$

For graph convolution based methods, InfoNCE proposed in [14] is utilized as the optimization target. From the perspective of instance discrimination, the contrastive loss is a function whose value is low when a query q is similar to its positive key k_+ and dissimilar to all other keys (negative keys). In most cases, contrastive learning is often used in conjunction with data augmentation, but data augmentation in graph is still a open problem. In this paper, we simply take input features and the smoothed result as positive pairs. For each node $\mathbf{h}_i \in \mathbf{H}$, $\hat{\boldsymbol{x}}_i \in \hat{\mathbf{X}}$ is the only positive key and all the other $\hat{\boldsymbol{x}}_j \in \hat{\mathbf{X}}, i \neq j$ are negative keys. Therefore, we can have the InfoNCE loss with respect to instance discrimination as:

$$Q_f = -\sum_{i=1}^{N} \log \frac{exp(\boldsymbol{h}_i^T \hat{\boldsymbol{x}}_i/\tau)}{\sum_{j=0}^{K} exp(\boldsymbol{h}_i^T \hat{\boldsymbol{x}}_j/\tau)} \tag{22}$$

where τ is a temperature hyperparameter.

Algorithm 2 Propagation

```
Input: Normalized ajacency matrix $\hat{\mathbf{A}}$; Initial embedding $\hat{\mathbf{X}}$;
    Type of graph filter $gf$;
Output: Enhanced embedding $\mathbf{H}$
 1: Order of Expansion $k$ = 5
 2: Laplacian matrix $\hat{\mathbf{L}} = \mathbf{I} - \hat{\mathbf{A}}$ $\hat{\mathbf{X}}^{(0)} = \hat{\mathbf{X}}$
 3: if $gf \in$ [Heat kernel, Gaussian] then
 4:     $\bar{\mathbf{X}}^{(1)} = \bar{\mathbf{L}}\mathbf{X}$
 5:     $\mathbf{H} = B(0)\hat{\mathbf{X}}^{(0)} - 2B(1)\hat{\mathbf{X}}^{(1)}$
 6:     for $i = 2$ to $k$ do
 7:         $\bar{\mathbf{X}}^{(i)} = 2\hat{\mathbf{L}}\hat{\mathbf{X}}^{(i-1)} - \hat{\mathbf{X}}^{(i-2)}$
 8:         $\mathbf{H} = \mathbf{H} + (-1)^i B(i)\hat{\mathbf{X}}^{(i)}$
 9:     end for
10: else if $gf$ is PPR then
11:     $\mathbf{H} = \alpha\bar{\mathbf{X}}^{(0)}$
12:     for $i = 1$ to $k$ do
13:         $\hat{\mathbf{X}}^{(i)} = (1-\alpha)\hat{\mathbf{A}}\hat{\mathbf{X}}^{(i-1)}$
14:         $\mathbf{H} = \mathbf{H} + \hat{\mathbf{X}}^{(i)}$
15:     end for
16: else
17:     $\mathbf{H} = Normalize(\hat{\mathbf{A}}\sigma(\mathbf{D}^{-1}))\mathbf{X}$
18: end if
19: return $\mathbf{H}$
```

3.3.3 Automatic Searching

Now that search space and unsupervised loss functions have been defined, we employ automatic machine learning to search for the best combination of graph filters and hyperparameters. As the loss functions are often non-convex and it is difficult to get the derivative and gradients, hyperparameter optimization in machine learning is generally considered as a problem of black-box optimization. Bayesian optimization [32] is a common and powerful method to tackle such cases. The basic idea of Bayesian optimization is to use the Bayesian rule to estimate the posterior distribution of the objective function based on dataset, and then select the next sampling hyperparameter combination according to the distribution. It aims to find the parameters that achieve the most improvement by optimizing the objective function, or loss function. In the implementation, we employ Optuna with Bayesian optimization as the backend AutoML framework for automatic searching. As shown in Algorithm 1, AutoProNE explores better filters and parameters based on previous attempts and losses.

4 Analysis and Discussions

We provide analyses of the impact of graph filters on the expressiveness of graph representations.

4.1 Insight into graph filters

4.1.1 PPR as a low-pass filter

PPR is defined in the spatial domain. Intuitively, through random walk, a node can capture the information of both direct and distant neighbors. The probability distribution of random walk with restart converges to a stationary distribution:

$$\begin{aligned} \mathbf{A}_p &= \alpha(\mathbf{I}_n - (1-\alpha)\mathbf{A}\mathbf{D}^{-1})^{-1} \\ &= \alpha(\alpha\mathbf{I}_n - (1-\alpha)\hat{\mathbf{L}})^{-1} \\ &= \mathbf{U}\mathrm{diag}(\frac{\alpha}{(1-\alpha)\lambda_i + \alpha})\mathbf{U}^T \end{aligned} \tag{23}$$

In the frequency domain, PPR kernel can be written as $f_p(\lambda) = \frac{\alpha}{(1-\alpha)\lambda+\alpha}$ and is a low-pass filter. This equals to our intuition that random walk usually assigns higher weights to low-order neighbors.

4.1.2 Spectral properties of graph filters

Generally, for the convenience of writing and without loss of generality, we denote a general graph filter matrix with $\mathcal{L}$, where $\mathbf{A}_p, \mathbf{L}_h$ and $\mathbf{L}_q$ are special cases of $\mathcal{L}$. Let λ_i be the i^{th} eigenvalue of $\mathcal{L}$, adjacency matrix $\mathcal{A} = \mathbf{I}_n - \mathcal{L} = \mathbf{I}_n - \mathbf{U}\mathrm{diag}(\lambda_1, ..., \lambda_n)\mathbf{U}^T$.

GCN in fact works like a low-pass filter from the spectral domain. The propagation of node features by multiplying the (augmented) adjacency matrix $\hat{\mathcal{A}}$ corresponds to applying graph filter $g(\lambda) = 1 - \lambda$. The eigenvalues of $\mathcal{A}$ lie on the interval [0, 2] and it has been proved in [15] that λ_{max}, the largest eigenvalue of $\mathcal{A}$ and $\hat{\lambda}_{max}$, the largest eigenvalue of $\hat{\mathcal{A}}$, satisfies:

$$0 < \hat{\lambda}_{\max} < \lambda_{\max} \leq 2 \tag{24}$$

Thus $g(\lambda) = 1 - \lambda$ resembles a band-stop filter. The graph filter of GCN can be described as $g(\lambda_i) = (1 - \lambda_i)^K$, where $K = 1$ in one GCN layer and can be any positive integer in Simplified GCN. The spectrum of $\hat{\mathcal{A}}^K$ actually works as a low-pass-type filter for $K > 1$ [33].

PPR and Heat kernel work as a low-pass filter. α and θ adjust the pass rate. Gaussian kernel can be generally considered as a band-pass filter. As μ determines the only peak, by adjusting μ and θ, we can implement different types of filter that passes frequencies within a certain range: $\mu \leq 0$ as low-pass; $\mu \geq \hat{\lambda}_{\max}$ as high-pass; $0 < \mu < \hat{\lambda}_{\max}$ as band-pass [28].

In the spectral domain, the graph filter extracts features in a certain band of the network. Generally, the true features are concentrated in a certain frequency band and the noises conform to uniform or Gaussian distribution. Thus graph filter helps preserve the intrinsic information and reduce the noise. In the spatial domain, the graph filters aggregate the information of both direct and distant neighborhoods into the node embedding. Therefore, the features are properly smoothed and the learned embedding can be more expressive.

4.1.3 Spatial Property of SR

The signal rescaling function is inspired by the fact that many graph convolutional networks are actually performing signal rescaling. In many networks, there exist nodes of very high degrees. For example, the average degree of nodes in BlogCatalog is 32, but the maximum degree is 1,996; DBLP's average node degree is only 2.5, but the maximum degree is also up to 90. The class of most nodes is not determined by its neighbors of a high degree, but neighbors with a lower degree. As a result, attenuating node signals of a high degree helps improve embedding performance in some structure information dominated networks.

4.1.4 *Visualization of Graph Filters*

Fig 2 exhibits the visualization results of the functions of different filters on Karate Clubs dataset. Compared to merely row normalization, as we can see from the figures, PPR, HeatKernel and Gaussian kernel enlarge the receptive field and assign different weights to neighbors and also pay attention to nonneighbor nodes.They do show different forms when capturing the characteristics of higher-order neighbors. In addition, the three filters assign high weights to the node itself even without adding self-loops. Signal rescaling adjusts weights according to the node degrees of neighbors.

Fig 2 shows the impact of the hyperparameters on filters. And it can be observed that PPR, HeatKernel and Gaussian Kernel are sensitive to hyperparameters, which mainly determine how a node aggregates the features of its neighbors. For PPR, the bigger restart probability α, the more attention a node pays to neighbor nodes. Gaussian and heat kernel are more complex in the spatial domain.

4.2 Connection with graph partition

Graph partition aims to reduce a graph into a set of smaller graphs by partitioning its set of nodes into mutually exclusive groups. In many cases, graphs have good locality and a node often tends to have similar features and be in the same class with its neighbors. The assumption is also the basis of many algorithms like *DeepWalk*, *LINE* and *GCN*. Therefore, the locality and connectivity of a graph are highly related to the effectiveness of the *Propagation* in our model. Luckily, high-order Cheeger's inequality [34], [35] suggests that eigenvalues of Laplacian matrix are closely associated with a graph's locality and connectivity.

The effect of graph partition can be measured by *graph conductance*(a.k.a. Cheeger constant). For a node subset $S \subseteq \mathcal{V}$, the volume of S is $vol(S) = \sum_{x \in S} d(x)$, $d(x)$ is the degree of node x. The *edge boundary* of S is $e(S) = \{(x, y) \in \mathcal{E} | x \in S, y \notin S\}$. Then the conductance of S is defined to be:

$$\Phi(S) = \frac{|e(S)|}{\min(vol(S), vol(\bar{S}))}$$

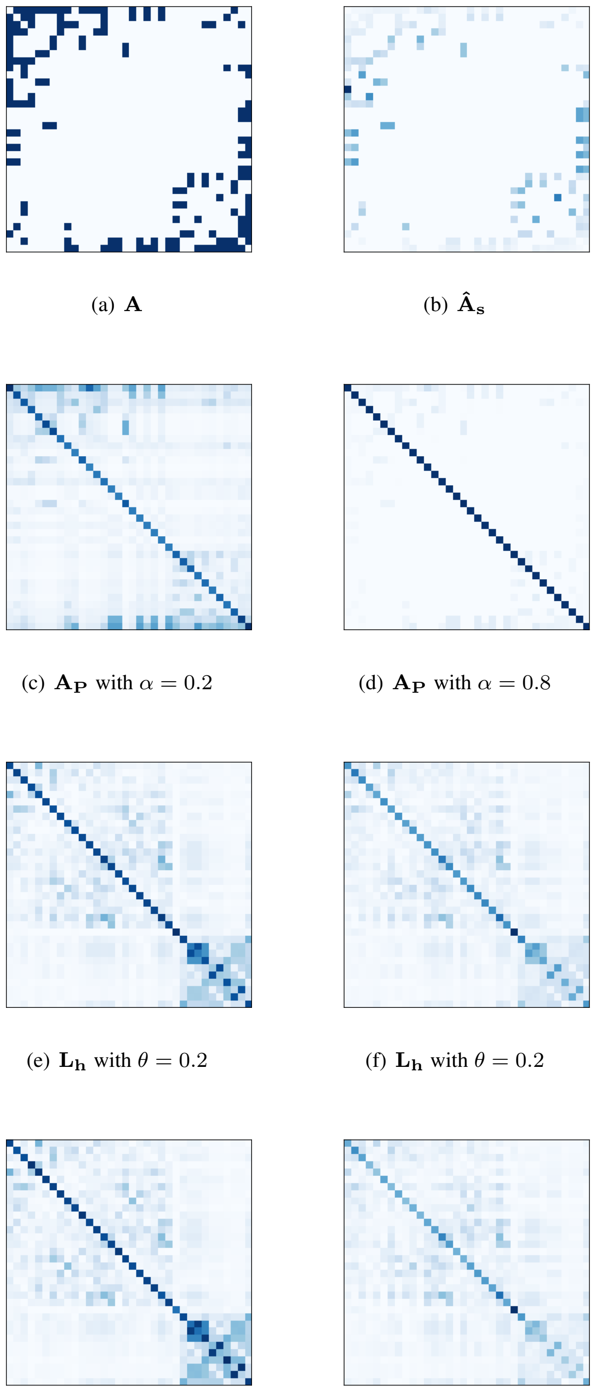
To measure the conductance of the whole graph $\mathcal{G}$ when partitioned into k parts, let $S_1, ..., S_k \subseteq \mathcal{V}$ and $S_i \cap S_j = \emptyset$ if $i \neq j$. The k-way Cheeger constant is defined as:

$$\rho_{\mathcal{G}}(k) = min\{max\{\Phi(S_i), i = 1, ..., k\}\}$$

$\rho_{\mathcal{G}}(k)$ is positively correlated to graph connectivity. if a graph has higher conductance(bigger $\rho_{\mathcal{G}}(k)$), it is better connected. High-order Cheeger equality bridges the gap between graph partition and graph spectrum and shows that the eigenvalues of Laplacian matrix control the bounds of k-way Cheeger constant as follows:

$$\frac{\lambda_k}{2} \leq \rho_{\mathcal{G}}(k) \leq O(k^2)\sqrt{\lambda_k} \tag{25}$$

In spectral graph theory, the number of connected components in an undirected graph is equal to the multiplicity of the eigenvalue zero in graph Laplacian.



(a) $\mathbf{A}$ (b) $\hat{\mathbf{A}}_{\mathbf{s}}$

(c) $\mathbf{A_P}$ with $\alpha = 0.2$ (d) $\mathbf{A_P}$ with $\alpha = 0.8$

(e) $\mathbf{L_h}$ with $\theta = 0.2$ (f) $\mathbf{L_h}$ with $\theta = 0.2$

(g) $\mathbf{L_g}$ with $\mu = 0.2$ and $\theta = 1.0$ (h) $\mathbf{L_g}$ with $\mu = -1$ and $\theta = 2$

Fig. 2. Visualization of adjacency matrix $\mathbf{A}$, row normalized adjacency matrix with self-loops $\hat{\mathbf{A}}_{\mathbf{rw}}$, PPR matrix $\mathbf{A_P}$, Gaussian filter matrix $\mathbf{L_g}$, Heat kernel filter matrix $\mathbf{L_h}$ and signal rescaling matrix $\mathbf{A}_r$ on dataset Karate Clubs. The darker the color means the higher the weight of the node. The weight is between [0, 1].

TABLE 2
Statistics of datasets.

Dataset	Nodes	Edges	Classes	Features	Degree
BlogCatalog	10,312	333,983	39	-	32.4
PPI	3,890	76,584	50	-	19.7
Wiki	4,777	184,812	40	-	38.7
DBLP	51,264	127,968	60	-	2.5
Youtube	1,138,499	2,990,443	47	-	2.6
Cora	2,708	5,429	7	1,433	2.0
Citeseer	3,327	4,732	6	3,703	1.4
Pubmed	19,717	44,338	3	500	2.2

Eq 25 indicates that we can increase(decrease) the graph conductance $\rho_{\mathcal{G}}(k)$ by increasing(decreasing) the lower bound λ_k. For example, low-pass filters increases λ_k for small k and decreases eigenvalues for big k. As a result, different parts become more isolated when the graph is partitioned into different groups, thus improving the locality of the graph and benefiting node classification. In practical applications, different types of graph filters can be selected to fit the characteristics of graphs.

4.3 Complexity

The computation of graph filters in Table 1 can be efficiently executed in a recurrent manner. Let $n = |\mathcal{V}|, m = |\mathcal{E}|$. With Taylor and Chebyshev expansion, we only apply repeated Sparse Matrix-Matrix multiplication (SPMM) between a sparse $n \times n$ matrix and a dense $n \times d$ feature matrix with time complexity $O(md)$ and avoid explicit eigendecomposition with complexity $O(n^3)$. In addition, Intel Math Kernel Library (MKL) provides efficient SPMM operators which can handle graph of millions or even billions of nodes [36]. This makes it possible for AutoProNE to handle large-scale graph data.

For Taylor expansion, we denote $\bar{\mathbf{X}}^{(i)} = \theta_i \hat{\mathbf{X}}^{(\mathbf{i})}, \hat{\mathbf{X}}^{(\mathbf{i})} = \hat{\mathbf{A}}\hat{\mathbf{X}}^{(i-1)}$ with $\hat{\mathbf{X}}^{(0)} = \hat{\mathbf{X}}$. For Chebyshev expansion, we denote $\bar{\mathbf{X}}^{(i+1)} = c_i(\theta)\hat{\mathbf{X}}^{(i)}, \hat{\mathbf{X}}^{(i)} = 2\hat{\mathbf{L}}\hat{\mathbf{X}}^{(\mathbf{i-1})} - \hat{\mathbf{X}}^{(\mathbf{i-2})}$ with $\hat{\mathbf{X}}^{(0)} = \hat{\mathbf{X}}$. As $\hat{\mathbf{L}}$ and $\hat{\mathbf{A}}$ are both sparse matrix, the complexity of *Propagation* is $O(kdm)$. The dimension reduction (SVD) on small matrix is $O(nd^2)$. And the complexity of computing Infomax and InfoNCE loss is also $O(nd^2)$ as it only involves the element-wise multiplication of matrix. All together, the complexity of AutoProNE is $O(nd^2 + kdm)$.

5 Experiments

We conduct extensive experiments to evaluate the effectiveness and efficiency of the proposed AutoProNE framework. Specifically, we examine to what extent AutoProNE can improve both shallow network embedding methods (without input feature matrix) and graph neural networks (with input feature matrix).

5.1 Experiment Setup

5.1.1 Datasets.

We use eight datasets that are commonly used for evaluating graph representation learning, 6 of which are relatively small scale but are widely used in graph representation learning literature, including BlogCatalog, PPI, Wiki, Cora, Citeseer and Pubmed. DBLP and Youtube are relatively large scale networks. Table 2 lists their statistics.

We use the following five datasets without input node features.

- **BlogCatalog** [37] is a network of social relationships of online bloggers. The vertex labels represent the interests of the bloggers.
- **PPI** [38] is a subgraph of the PPI network for Homo Sapiens. The vertex labels are obtained from the hallmark gene sets and represent biological states.
- **DBLP** [39] is an academic citation network where authors are treated as nodes and their dominant conferences as labels.
- **Wiki** [40] is a word co-occurrence network in part of the Wikipedia dump. Node labels are the Part-of-Speech tags.
- **Youtube** [37] is a video-sharing website that allows users to upload, view, rate, share, add to their favorites, report, comment on videos. The labels represent groups of viewers that enjoy common video genres.

We consider the following three datasets with input node features.

- **Cora** [41] is a paper citation network. Each publication is associated with a word vector indicating the absence/presence.
- **Citeseer** [41] is also a paper citation network. Each node has a human-annotated topic as its label and content-based features.
- **Pubmed** [42] contains 19717 diabetes-related publications. Each paper in Pubmed is represented by a term frequency-inverse document frequency vector.

5.1.2 Baselines

We compare with several state-of-the-art methods including:

- **DeepWalk** [4] DeepWalk generalizes language model to graph learning. For each vertex, truncated random walks starting from the vertex are used to obtain the contextual information.
- **LINE** [5] LINE preserves the first-order and second-order proximity between vertexes. And we use LINE with the second order proximity.
- **Node2Vec** [6] Node2vec designs a second order random walk strategy to sample the neighborhood nodes, which interpolates between breadth-first sampling and depth-first sampling.
- **HOPE** [7] HOPE preserves high-order proximities of graphs and capable of capturing the asymmetric transitivity using matrix factorization.
- **NetMF** [8] NetMF unifies skip-gram based methods as matrix factorization.
- **ProNE** [16] ProNE is a fast network embedding method including sparse matrix factorization and spectral propagation. And we use **ProNE(SMF)** to represent ProNE only with sparse matrix factorization.
- **GraphSAGE** [10] GraphSAGE is an inductive GNN framework generating node embeddings by sampling and aggregating features from a node's local neighborhood.
- **DGI** [11] Deep Graph Infomax (DGI) maximizes mutual information and classifies local-global pairs and negative-sampled counterparts.
- **GCN** [1] Graph convolution network (GCN) simplifies graph convolutions by restricting the graph filters to operate in an 1-step neighborhood around each node.
- **GAT** [9] Graph attention networks adopt attention mechanisms to learn the relative weights between two connected nodes.

Stacked with the proposed graph filters block, we evaluate how the algorithm performance can be improved.

5.1.3 Implementation Details.

For a fair comparison, we set the dimension of embedding $d = 128$ for all network embedding methods. For all the other parameters,

TABLE 3
Micro-F1 results of node classification by different algorithms w/ and w/o AutoProNE. Ratio indicates the percentage of labeled data. Numbers in the brackets indicate performance improvements and all are statistically significant (p-value$\ll$0.01; t-test).

Datasets	Ratio	DeepWalk	ProDeepWalk	LINE	ProLINE	node2vec	ProNode2vec	NetMF	ProNetMF	HOPE	ProHOPE
BlogCatalog	0.1	35.6	36.2 (+1.7%)	30.3	32.3 (+6.6%)	35.7	35.7 (0.0%)	35.6	36.8 (+3.4%)	30.4	33.5 (+10.2%)
	0.5	40.5	41.8 (+3.2%)	37.1	38.3 (+3.2%)	40.6	41.5 (+2.2%)	41.1	41.9 (+2.0%)	34.0	37.4 (+10.0%)
	0.9	42.3	44.2 (+4.5%)	39.6	40.5 (+2.3%)	41.8	43.9 (+5.0%)	41.6	42.3 (+1.7%)	35.3	38.3 (+8.5%)
PPI	0.1	16.7	17.6 (+5.4%)	12.5	17.0 (+34.9%)	16.1	17.5 (+8.7%)	17.9	18.0 (+0.6%)	16.0	17.3 (+8.1%)
	0.5	21.6	24.7 (+14.3%)	16.4	23.7 (+44.5%)	20.6	24.1 (+17.0%)	23.1	24.6 (+6.5%)	21.0	23.3 (+10.9%)
	0.9	24.0	27.0 (+12.5%)	19.5	26.2 (+34.4%)	23.1	25.7 (+11.2%)	25.5	26.5 (+3.9%)	23.2	24.9 (+7.3%)
Wiki	0.1	43.3	44.5 (+2.8%)	41.8	45.8 (+9.6%)	44.8	44.2 (-1.3%)	45.7	45.9 (+0.5%)	48.8	48.0 (+2.4%)
	0.5	49.2	50.0 (+1.6%)	52.5	53.2 (+1.3%)	51.1	50.9 (-0.3%)	50.1	50.9 (+1.6%)	53.1	52.8 (-0.5%)
	0.9	50.0	51.4 (+2.8%)	54.7	55.0 (+0.5%)	52.8	52.5 (-0.5%)	50.7	51.9 (+2.4%)	53.1	54.3 (+0.4%)
DBLP	0.01	51.5	55.8 (+8.3%)	49.7	52.4 (+5.4%)	53.4	57.8 (+8.2%)	51.5	52.9 (+2.7%)	-	-
	0.05	58.1	59.0 (+1.5%)	54.9	56.2 (+2.4%)	58.3	60.0 (+2.9%)	57.1	59.5 (+4.2%)	-	-
	0.09	59.4	59.9 (+0.9%)	56.3	57.0 (+1.2%)	59.5	60.6 (+1.8%)	57.9	60.2 (+4.0%)	-	-
Youtube	0.01	38.2	39.2 (+2.6%)	33.2	39.8 (+19.8%)	38.2	39.7 (+3.9%)	-	-	-	-
	0.05	41.6	44.7 (+6.0%)	36.2	43.5 (+20.1%)	40.0	45.3 (+12.2%)	-	-	-	-
	0.09	42.8	46.2 (+7.9%)	38.3	45.9 (+19.8%)	43.0	47.1 (+9.5%)	-	-	-	-

TABLE 4
Accuracy results of node classification by unsupervised GNNs. Significant test (p-value$\ll$0.01; t-test).

	Dataset	Cora	Pubmed	Citeseer
Semi-supervised	GCN	81.5	79.0	70.3
	GAT	83.0 $\pm$ 0.7	79.0 $\pm$ 0.3	72.5 $\pm$ 0.7
Unsupervised	DGI	82.0 $\pm$ 0.1	77.1 $\pm$ 0.1	71.7 $\pm$ 0.2
	ProDGI	82.9 $\pm$ 0.2	81.0 $\pm$ 0.1	70.8 $\pm$ 0.2
	GraphSAGE	77.2 $\pm$ 0.1	78.0 $\pm$ 0.1	61.2 $\pm$ 0.1
	ProSAGE	78.1 $\pm$ 0.2	79.5 $\pm$ 0.1	62.1 $\pm$ 0.2

we follow the authors' original setup and have the following settings: For DeepWalk, windows size m = 10, #walks per node r = 80, walk length t = 40; For Node2Vec, window size m = 10, #walks per node r = 80, walk length t = 40, p, q are searched over $\{0.25, 0.50, 1, 2, 4\}$. For LINE, #negative-samples k = 5 and total sampling budget $T = r \times t \times |V|$. For HOPE, β is set to be 0.01. For NetMF, window size r = 5, rank = 256, #negative-samples k = 5. For DeepWalk, LINE and Node2Vec, we use the official code provided by the original authors and for NetMF and HOPE, we use the code implemented in CogDL.

To further validate the effectiveness of AutoProNE, we also implement AutoProNE on unsupervised convolution-based methods DGI [11] and unsupervised GraphSAGE [10], and compare with semi-supervised GCN [1] and GAT [9]. All parameters are set the same as in the authors' original settings. We use the official code provided by the original authors of DGI and unsupervisd GraphSAGE code implemented in CogDL.

For AutoProNE, each filter can only be selected once. The term number of Taylor and Chebyshev expansion k is set to be 5. For PPR, α is searched between [0.1, 0.9]. For Heat kernel, θ in [0.1, 0.9]. For Gaussian Kernel, μ in [0, 2], θ in [0.2, 1.5].

Evaluation For non-convolution based methods, we follow the same experimental settings used in baseline works [4], [5], [6], [16] . We randomly sample different percentages of labeled nodes to train a liblinear classifier and use the remaining for testing. The training ratio for small datasets(PPI, Wiki, BlogCatalog) is 0.1/0.5/0.9, and 0.01/0.05/0.09 for relatively big datasets(DBLP and Youtube). The remaining is for predicting. We repeat the training and predicting 10 times and report the average Micro-F1 for all methods. Analogous results also hold for Macro-F1, which are not shown due to space constraints. For unsupervised convolutional methods, we train a multi-layer perception with a fixed train/valid/test data splitting the same as in [41]. We repeat it 50 times and report the average accuracy for all methods.

5.2 Results

5.2.1 *Overall Performance.*

Tables 3 lists the node classification performance based on the embeddings learned by different network embedding algorithms with and without AutoProNE. We test different ratios (0.1, 0.5, 0.9) of labeled data for node classification following existing work [4], [5], [6] to train a liblinear classifier and repeat the training and predicting for ten times and report the average Micro-F1 for all methods.

We observe that the performance of all the base algorithms can be significantly improved by the AutoProNE framework and also the improvements are statistically significant. For DeepWalk, LINE, node2vec, NetMF, and HOPE, the improvements brought by AutoProNE are up to 14.3%, 44.5%, 17%, 6.5%, and 10.9%, respectively. On average, LINE benefits more from AutoProNE as it is in nature an embedding method without incorporating high-order structural information.

Table 4 reports the results of unsupervised GraphSAGE and DGI as well as the AutoProNE version of them. As a reference point, we also list the performance of two popular semi-supervised GNNs—GCN and GAT. We observe that the unsupervised AutoProNE framework can help improve the performance of both DGI and GraphSAGE in most cases. This suggests that by automated usage of graph filters, the simple AutoProNE strategy is capable of enhancing graph representation learning with or without input

TABLE 5
Results of base embedding methods with different graph filters. Ratio=0.5 for BlogCatalog, PPI, and Wiki; 0.05 for DBLP and Youtube.

Dataset	Type	DeepWalk	LINE	node2vec	NetMF	HOPE
BlogCatalog	Heat	41.4	38.3	41.0	41.6	34.6
	PPR	41.7	38.8	41.3	41.8	35.6
	Gaussian	41.4	38.4	41.1	41.3	37.5
	SR	41.9	38.7	40.9	41.6	34.5
	AutoProNE	41.8	38.3	41.5	41.9	37.4
PPI	Heat	23.4	21.1	22.8	24.0	21.7
	PPR	23.9	22.7	23.6	24.3	22.3
	Gaussian	23.2	21.3	22.8	23.7	21.1
	SR	24.5	23.0	24.8	25.0	23.0
	AutoProNE	24.7	23.7	24.1	24.6	23.3
Wiki	Heat	48.2	52.8	50.5	47.4	53.1
	PPR	48.4	52.6	49.9	48.1	53.2
	Gaussian	48.2	52.6	50.3	47.2	53.0
	SR	49.0	54.1	52.2	50.7	53.0
	AutoProNE	50.0	53.2	50.9	50.9	52.8
DBLP	Heat	58.8	54.7	59.4	58.8	-
	PPR	59.0	55.4	59.7	59.0	-
	Gaussian	58.9	54.7	59.7	58.5	-
	SR	58.7	54.1	59.7	58.9	-
	AutoProNE	59.0	56.2	60.0	59.5	-
Youtube	Heat	44.5	42.7	44.7	-	-
	PPR	44.6	43.5	45.1	-	-
	Gaussian	44.3	40.5	44.3	-	-
	SR	44.5	43.0	45.1	-	-
	AutoProNE	44.7	43.5	45.3	-	-

TABLE 6
Results of base GNNs with different graph filters.

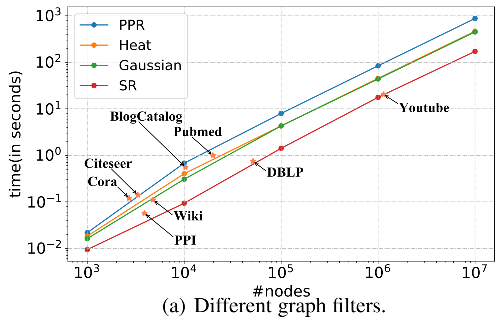
Dataset	Type	Cora	Citeseer	Pubmed
DGI	Heat	82.0	71.8	77.5
	PPR	64.2	70,1	77.4
	Gaussian	82.9	71.4	80.7
	SR	13.1	70.1	77.2
	AutoProNE	82.9	70.8	81.0
GraphSAGE	Heat	76.5	61.7	77.4
	PPR	62.1	62.0	77.6
	Gaussian	77.8	62.3	76.6
	SR	16.5	62.7	77.1
	AutoProNE	78.1	62.2	79.5

node features. With the help of AutoProNE, DGI can even yield better performance than the end-to-end semi-supervised GCN and GAT models on Pubmed.

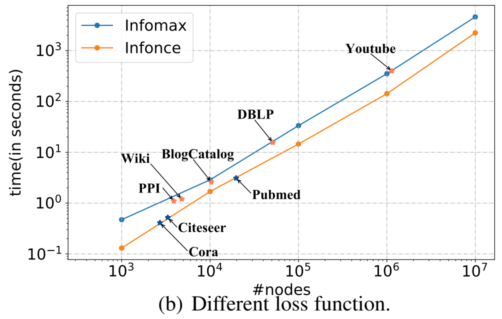
Finally, we observe that the AutoProNE prefers the newly proposed signal rescaling (SR) filter for BlogCatalog, PPI, Wiki, DBLP and Youtube, while it tends to favor Gaussian kernel for Cora, Citeseer, and Pubmed. We leave the reason behind this difference for future work.

5.2.2 *The Role of AutoML.*

Tables 5 and 6 report the performance of base methods with each of the four graph filters in AutoProNE's search space. We



(a) Different graph filters.



(b) Different loss function.

Fig. 3. AutoProNE's Scalability w.r.t. network volume. Running time as #node grows with node degree fixed to 10. As the network size increases, the time cost of both graph filters and computing loss also grows linearly.

can observe that different filters have unique impacts on the performance across datasets. For example, Signal Rescaling(SR) and PPR perform relatively better than other filters in dataset PPI, BlogCatalog and Wiki, but for Cora, Citeseer and Pubmed, Gaussian filter exhibits better performance. And the tables show that the performance of AutoProNE is equal to the best result of one single filter, which may imply that AutoProNE finally picks this filter, or our model yields better performance than any single filter, which means AutoProNE learns a better combination of graph filters. These all suggest the need for AutoML to select the best filter(s) for each dataset. The proposed AutoProNE strategy consistently and automatically picks the optimal filter(s) and parameters in an unsupervised manner.

The graph filter used for the embedding smoothing in ProNE [16] is a modified 2nd-order Gaussian kernel. For simplicity, the 1st-order Gaussian kernel is covered in AutoProNE's search space. Table 7 reports the performance for ProNE(SMF) enhanced by ProNE graph filter and AutoProNE respectively. We can see that the automated search strategy empowers AutoProNE to generate better performance than ProNE in most cases, further demonstrating the effectiveness of using AutoML for smoothing graph representations.

From the experiment results of AutoML searching, we also have some interesting findings. PPR, Heat kernel and Gaussian kernel perform better if high-order neighbor information matters such as Cora, because SR only aggregates 1st-order neighbors. Besides, low-order methods(like LINE) may benefit more because the original embeddings fail to incorporate abundant neighborhood information.

TABLE 7
Results of ProNE and AutoProNE. *With ProNE* and *With AutoProNE* mean the result of ProNE(SMF) improved by the graph filter of ProNE and AutoProNE respectively.

Method	BlogCatalog			PPI			Wiki			DBLP			Youtube		
Ratio	0.1	0.5	0.9	0.1	0.5	0.9	0.1	0.5	0.9	0.01	0.05	0.09	0.01	0.05	0.09
ProNE(SMF)	33.0	37.8	39.1	15.1	22.0	24.5	47.8	54.5	55.6	46.7	54.0	55.6	36.4	41.4	42.5
With ProNE	36.4	40.9	42.2	17.5	24.0	26.5	48.0	54.6	56.0	46.7	55.2	56.3	37.6	42.9	43.9
With AutoProNE	35.8	41.0	42.2	17.7	24.3	26.5	48.6	55.5	56.8	50.0	56.1	57.5	38.7	43.7	44.6

TABLE 8
Efficiency of AutoProNE (seconds). SMF stands for ProNE(SMF).

Dataset	DeepWalk	LINE	node2vec	SMF	NetMF	HOPE	AutoProNE
PPI	272	70	716	2	10	12	**+11** (4.0%)
Wiki	494	87	819	4	23	17	**+12** (2.4%)
BlogCatalog	1,231	185	2,809	12	144	136	**+29** (2.3%)
DBLP	3,825	1,204	4,749	15	186	-	**+100** (2.6%)
Youtube	68,272	5,890	30,218	302	-	-	**+3,213** (4.7%)

TABLE 9
Efficiency of AutoProNE (seconds).

Dataset	DGI	GraphSAGE	AutoProNE
Cora	341	118	**+15** (4.1%)
Citeseer	490	131	**+21** (4.2%)
Pubmed	4,863	1,254	**+183** (3.7%)

5.2.3 *Efficiency and Scalability.*

We follow the common practice for efficiency evaluation by the wall-clock time and AutoProNE 's scalability is analyzed by the time cost in multiple-scale networks [5].

Tables 8 and 9 report the extra running time (seconds) when stacked with AutoProNE for 100 searching iterations in 10 threads. Note that AutoProNE is a dataset specific and base agnostic framework. The percentage of extra running time in terms of DeepWalk and DGI is also reported inside the brackets, respectively.

We can observe that AutoProNE is very efficient compared to base methods. Overall, AutoProNE costs only 2.3–4.7% of DeepWalk's or DGI's running time. Take the Youtube graph as an example, it takes DeepWalk 68,272 seconds ($\sim$19 hours) for generating the embeddings of its 1,000,000+ nodes. However, AutoProNE only needs 4.7% of its time to offer 2.6–7.9% performance improvements (Cf. Table 3). For convolutional methods, GNNs are usually less efficient for large scale datasets. This suggests that AutoProNE will take less of the additional percentage of time to achieve improvement.

We use synthetic networks to demonstrate the scalability of AutoProNE. We generate random regular graphs with a fixed node degree as 10 and the number of nodes ranging between 1,000 and 10,000,000. In addition, we also add the running time on each dataset with the heat kernel and the corresponding loss.

AutoProNE is ideal for parallel implementation. The computation of our model is mainly spent on iteratively selecting different filters and hyperparameters to evaluate the effectiveness. Therefore, running multi-progresses will speed up the searching of AutoML and achieves great efficiency.

Table 8 shows that ProNE(SMF) is a fast network embedding method. With ProNE(SMF) as the base embedding method, AutoProNE works as a general network embedding method with high efficiency and also achieves comparable performance, as shown in Table 7.

6 Related Work

6.1 Network Embedding

Network embedding has been extensively studied by machine learning communities in the past few years and aims to train low dimension vectors that are available for a wide range of downstream tasks.

The recent emergency of network embedding research begins when skip-gram model [43], [44], which is originally used in word representation learning and network mining, is applied to derive the embedding of nodes in networks. DeepWalk [4] and Node2Vec [6] employ random walks to explore the network structure and LINE [5] takes a similar idea with an explicit objective function by setting the walk length as one. These random walk based methods can be understood as implicit matrix factorization [8].

The other explicit matrix factorization based network embedding methods have also been proposed. GraRep [45] directly factorizes different k-order proximity matrices and HOPE [7] proposes to use generalized SVD to preserve the asymmetric transitivity in directed networks. [46] also proposes a framework to unify the aforementioned methods. ProNE [16] formalizes network embedding as sparse matrix factorization and preserves the distributional similarity. ProNE enhances the result of sparse matrix factorization with spectral propagation, which modulates the adjacency matrix mainly to incorporate the global properties in the spatial domain. AutoProNE generalizes the operation to be stacked with existing unsupervised representation learning algorithms in an automated way and improves their performance.

6.2 Graph Convolution Networks

Recently semi-supervised graph learning with graph neural networks, such as graph convolution networks (GCNs) [1], [47], [48], draws considerable attention. Various variants [49], [50], have also been designed to boost the performance. In GCNs, the convolution operation is defined in the spectral space and parametric filters are learned via back-propagation. [24], [51], [52], [53] replace the adjacency matrix in GCNs with graph filters (PPR and Heat kernel), which they called graph diffusion matrix, to combine the strengths of both spatial and spectral methods and the performance improves. They apply graph filters to semi-supervised learning in which graph filters are entangled with the model's training process. AutoProNE proposes a more flexible combination of graph filters in both spectral and spatial domains, which can filter the frequency in any specific band and better extract the intrinsic information and is model-agnostic. Besides, self-supervised learning [54] is emerging recently and contrastive learning shows great potential. Contrastive methods may employ a scoring function to evaluate the similarity of pairs of data. [14], [55] employ InfoNCE to maximize the gap between positive and negative pairs. One possible intuition behind this is that InfoNCE can shorten the distance between positive pairs, and make the distance between negative pairs as far as possible. [13] is based on classifying local-global pairs and negative-sampled pairs. And the ultimate goal is to maximize the local-global mutual information. [11], [56], [57], [58], [59], [60] apply contrastive methods in graph representation learning and achieves promising results. [56] contrasts the diffusion result of two graph filters, which also indicates that different filters captures different views of the graph. [12] employs contrastive coding to pre-train graph neural networks. AutoProNE also maximizes InfoNCE and local-global mutual information in AutoML loss for optimization.

6.3 Automated Machine Learning

With the arising of AutoML [61], [62], several works also apply it in graph representation learning. GraphNAS [63] and AutoGNN [64] aim to generate neural architectures of GNNs in spatial domain with recurrent neural network generator by using reinforcement learning with neural architecture search [65]. Both of them suffer from high computation costs to find the best model architecture for a given dataset. AutoProNE is more like a plug-and-play framework that can be applied to any graph node embeddings efficiently. Besides, Bayesian optimization [32], with the Gaussian process [66] as the underlying surrogate model, is a popular technique for finding the globally optimal solution of an optimization problem. And this technique is widely used especially in hyperparameter optimization. As AutoProNE mainly aims to search for the best hyperparameters for graph filters, Bayesian Optimization is the principle behind the searching. AutoProNE is an unsupervised and task-independent model that aims to pre-train general embeddings.

7 Conclusion

In this paper, we investigate the role of graph filters and propose an automated and unsupervised framework AutoProNE to generate improved graph embeddings for unsupervised graph representation learning. AutoProNE comprises four graph filters(PPR, HeatKernel, Gaussian Kernel and Signal Rescaling) and automatically searches for a combination of graph filters and corresponding hyperparameters for the given dataset. Specifically, AutoProNE operates on the adjacency matrix to enrich the context information of each node. It is a flexible and adaptive framework, as the graph filter set can simulate many kinds of filtering functions. It's also very efficient and costs only a little extra time to obtain the improvement in performance.

The developed method is model-agnostic and can be easily stacked with all unsupervised graph representation learning methods such as DeepWalk, LINE, node2vec, NetMF, and HOPE. On eight publicly available datasets, AutoProNE helps significantly improve the performance of various algorithms. In addition, AutoProNE can also enhance the performance of self-supervised/unsupervised GNN methods, e.g., DGI and GraphSage. We show that the self-supervised DGI model with the unsupervised AutoProNE can generate comparable or even better performance than semi-supervised end-to-end GNN methods, such as GCN and GAT. We have implemented AutoProNE in CogDL, an open-source graph learning library, to help more graph representation learning methods.

Acknowledgments

The work is supported by the National Key R&D Program of China (2018YFB1402600), NSFC for Distinguished Young Scholar (61825602), and NSFC (61836013).

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[2] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *KDD*, 2018.
[3] M. Ding, C. Zhou, Q. Chen, H. Yang, and J. Tang, "Cognitive graph for multi-hop reading comprehension at scale," in *ACL*, 2019.
[4] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *KDD*, 2014.
[5] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding." in *WWW*, 2015.
[6] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016.
[7] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *KDD*, 2016.
[8] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *WSDM*, 2018.
[9] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.
[10] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.
[11] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.
[12] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, "Gcc: Graph contrastive coding for graph neural network pre-training," in *KDD*, 2020.
[13] R. H. Devon, F. Alex, L.-M. Samuel, G. Karan, B. Phil, T. Adam, and B. Yoshua, "Learning deep representations by mutual information estimation and maximization," in *ICLR*, 2019.
[14] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *CVPR*, 2020.
[15] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML*, 2019.
[16] J. Zhang, Y. Dong, Y. Wang, J. Tang, and M. Ding, "Prone: fast and scalable network representation learning," in *IJCAI*, 2019.
[17] Y. Cen, Z. Hou, Y. Wang, Q. Chen, Y. Luo, X. Yao, A. Zeng, S. Guo, P. Zhang, G. Dai *et al.*, "Cogdl: An extensive toolkit for deep learning on graphs," *arXiv preprint arXiv:2103.00959*, 2021.
[18] B. Girault, A. Ortega, and S. S. Narayanan, "Irregularity-aware graph fourier transforms," *IEEE Transactions on Signal Processing*, vol. 66, no. 21, pp. 5746–5761, 2018.
[19] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs," *IEEE transactions on signal processing*, vol. 61, no. 7, pp. 1644–1656, 2013.
[20] 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*, vol. 30, no. 3, pp. 83–98, 2013.
[21] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *arXiv preprint arXiv:1908.00709*, 2019.
[22] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *KDD*, 2019.
[23] K. Tu, J. Ma, P. Cui, J. Pei, and W. Zhu, "Autone: Hyperparameter optimization for massive network embedding," in *KDD*, 2019.
[24] J. Klicpera, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," in *NIPS*, 2019, pp. 13 333–13 345.
[25] B. Xu, H. Shen, Q. Cao, K. Cen, and X. Cheng, "Graph convolutional networks using heat kernel for semi-supervised learning," in *IJCAI*, 2019.
[26] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.
[27] R. I. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete structures," in *ICML*, 2002.
[28] D. I. Shuman, B. Ricaud, and P. Vandergheynst, "Vertex-frequency analysis on graphs," *Applied and Computational Harmonic Analysis*, vol. 40, no. 2, pp. 260–291, 2016.
[29] L. C. Andrews, "Special functions of mathematics for engineers," vol. 49, 1998.

[30] C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng, “Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding,” in *ICLR*, 2020.

[31] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang, “Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization,” in *ICLR*, 2020.

[32] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” in *NIPS*, 2012.

[33] H. NT and T. Maehara, “Revisiting graph neural networks: All we have is low-pass filters,” *arXiv preprint arXiv:1905.09550*, 2019.

[34] J. R. Lee, S. O. Gharan, and L. Trevisan, “Multiway spectral partitioning and higher-order cheeger inequalities,” *JACM*, 2014.

[35] A. S. Bandeira, A. Singer, and D. A. Spielman, “A cheeger inequality for the graph connection laplacian,” *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 4, 2013.

[36] J. Qiu, L. Dhulipala, J. Tang, R. Peng, and C. Wang, “Lightne: A lightweight graph processing system for network embedding,” in *SIGMOD*, 2021.

[37] R. Zafarani and H. Liu, “Social computing data repository at asu,” 2009.

[38] B.-J. Breitkreutz, C. Stark, T. Reguly, L. Boucher, A. Breitkreutz, M. Livstone, R. Oughtred, D. H. Lackner, J. Bähler, V. Wood *et al.*, “The biogrid interaction database: 2008 update,” *Nucleic acids research*, vol. 36, no. suppl_1, pp. D637–D640, 2007.

[39] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, “Arnetminer: extraction and mining of academic social networks,” in *KDD*, 2008.

[40] M. Mahoney, “Large text compression benchmark,” *URL: http://www. mattmahoney. net/text/text. html*, 2009.

[41] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, “Automating the construction of internet portals with machine learning,” *Information Retrieval*, vol. 3, no. 2, pp. 127–163, 2000.

[42] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, “Collective classification in network data,” *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.

[44] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *NIPS*, 2013.

[45] S. Cao, W. Lu, and Q. Xu, “Grarep: Learning graph representations with global structural information,” in *CIKM*, 2015.

[46] T. Chen and Y. Sun, “Task-guided and path-augmented heterogeneous network embedding for author identification,” in *WSDM*, 2017.

[47] M. Henaff, J. Bruna, and Y. LeCun, “Deep convolutional networks on graph-structured data,” *arXiv preprint arXiv:1506.05163*, 2015.

[48] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *NIPS*, 2016.

[49] H. Chen, H. Yin, X. Sun, T. Chen, B. Gabrys, and K. Musial, “Multi-level graph convolutional networks for cross-platform anchor link prediction,” in *KDD*, 2020.

[50] H. Chen, H. Yin, T. Chen, Q. V. H. Nguyen, W.-C. Peng, and X. Li, “Exploiting centrality information with graph convolutions for network representation learning,” in *ICDE*, 2019.

[51] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, “Representation learning on graphs with jumping knowledge networks,” in *ICML*, 2018, pp. 5449–5458.

[52] J. Klicpera, A. Bojchevski, and S. Günnemann, “Predict then propagate: Graph neural networks meet personalized pagerank,” in *ICLR*, 2019.

[53] B. Jiang, D. Lin, J. Tang, and B. Luo, “Data representation and learning with graph diffusion-embedding networks,” in *CVPR*, 2019.

[54] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, “Self-supervised learning: Generative or contrastive,” *TKDE*, 2021.

[55] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” *arXiv preprint arXiv:2002.05709*, 2020.

[56] A. K. Sankararaman, S. De, Z. Xu, R. W. Huang, and T. Goldstein, “Contrastive multi-view representation learning on graphs,” *ICML*, 2020.

[57] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, “Deep graph contrastive representation learning,” *arXiv preprint arXiv:2006.04131*, 2020.

[58] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” in *NIPS*, 2020.

[59] Y. You, T. Chen, Y. Shen, and Z. Wang, “Graph contrastive learning automated,” in *ICML*, 2021.

[60] Y. Jiao, Y. Xiong, J. Zhang, Y. Zhang, T. Zhang, and Y. Zhu, “Sub-graph contrast for scalable self-supervised graph representation learning,” in *ICDM*, 2020.

[61] J. Bergstra, D. Yamins, and D. Cox, “Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures,” in *ICML*, 2013.

[62] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, “Towards automatically-tuned neural networks,” in *Workshop on Automatic Machine Learning*, 2016.

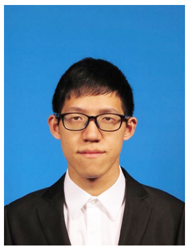
[63] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, “Graphnas: Graph neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1904.09981*, 2019.

[64] K. Zhou, Q. Song, X. Huang, and X. Hu, “Auto-gnn: Neural architecture search of graph neural networks,” *arXiv preprint arXiv:1909.03184*, 2019.

[65] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1611.01578*, 2016.

[66] J. Močkus, “On bayesian methods for seeking the extremum,” in *Optimization techniques IFIP technical conference*. Springer, 1975.

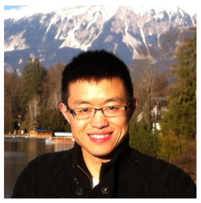
Zhenyu Hou is an undergraduate with the department of Computer Science and Technology, Tsinghua University. His main research interests include graph neural networks and representation learning.



Yukuo Cen is a PhD candidate in the Department of Computer Science and Technology, Tsinghua University. He got his bachelor degree in Computer Science and Technology from Tsinghua University. His research interests include social influence and graph embedding.

Yuxiao Dong received his Ph.D. in Computer Science from University of Notre Dame in 2017. He is a research scientist at Facebook AI, Seattle, was previously a senior researcher at Microsoft Research, Redmond. His research focuses on data mining, representation learning, and networks, with an emphasis on developing machine learning models to addressing problems in large-scale graph systems.

Jie Zhang Jie Zhang obtained his Bachelor’s degree in Mathematics and Physics from the Department of Physics at Tsinghua University in 2016, and received his master degree in Department of Computer Science and Technology, Tsinghua University in 2019. His research interests include graph neural networks, and data mining on graphs.



Jie Tang received the PhD degree from Tsinghua University. He is a professor in the Department of Computer Science and Technology, Tsinghua University. His main research interests include data mining, social network, and machine learning. He has published over 200 research papers in top international journals and conferences.