GraphAdaMix: Enhancing Node Representations with Graph Adaptive Mixtures

Da Sun Handason Tam
The Chinese University of Hong Kong

Siyue Xie
The Chinese University of Hong Kong

Wing Cheong Lau
The Chinese University of Hong Kong

Abstract

Graph Neural Networks (GNNs) are the current state-of-the-art models in learning node representations for many predictive tasks on graphs. Typically, GNNs reuses the same set of model parameters across all nodes in the graph to improve the training efficiency and exploit the translationally-invariant properties in many datasets. However, the parameter sharing scheme prevents GNNs from distinguishing two nodes having the same local structure and that the translation invariance property may not exhibit in real-world graphs. In this paper, we present Graph Adaptive Mixtures (GraphAdaMix), a novel approach for learning node representations in a graph by introducing multiple independent GNN models and a trainable mixture distribution for each node. GraphAdaMix can adapt to tasks with different settings. Specifically, for semi-supervised tasks, we optimize GraphAdaMix using the Expectation-Maximization (EM) algorithm, while in unsupervised settings, GraphAdaMix is trained following the paradigm of contrastive learning. We evaluate GraphAdaMix on ten benchmark datasets with extensive experiments. GraphAdaMix is demonstrated to consistently boost state-of-the-art GNN variants in semi-supervised and unsupervised node classification tasks. The code of GraphAdaMix is available online [1].

[1] `https://github.com/handasontam/GraphAdaMix`

Proceedings of the 25[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151.

1 INTRODUCTION

Recently, there has been a surge of academic interest in machine learning on graphs as graphs are ubiquitous in various real-world scenarios including social recommendations (Fan et al., 2019), traffic forecasting (Yu et al., 2017), COVID-19 pandemic forecasting (Kapoor et al., 2020), etc. In this paper, we focus on the task of transductive graph representation learning. It requires a model to learn discriminative embeddings for nodes in the graph, which can be applied to other downstream tasks such as node classifications (Kipf and Welling, 2016a) and link predictions (Kipf and Welling, 2016b).

Among all, Graph Neural Networks (GNNs) are a family of models that achieve state-of-the-art performance on graph representation learning. A key design ingredient of GNNs is that it uses parameter sharing (Chiang et al., 2019; Hamilton et al., 2017; Kipf and Welling, 2016a; Veličković et al., 2017) to improve the training efficiency. Such a scheme could be helpful and can be served as a good inductive bias, especially in computer vision, because it implicitly exploits the translationally-invariant properties of an image. However, this assumption may not hold in graphs. On one hand, as mentioned in You et al. (2019), the parameter sharing scheme used in typical GNNs limits its expressive power as they fail to distinguish between two nodes that have isomorphic neighborhoods, as shown in Figure 1. On the other hand, graphs equipped with different feature extractors may work better when nodes are with different contexts (e.g., locations, neighborhood information, etc.) (Hallac et al., 2015). One practical example is when the input is a social network where nodes represent persons and edges represent their social relationships. We might expect that different culture-specific features should be learned in different spatial locations due to cultural diversity. In this case, it is reasonable to relax the parameter sharing scheme. Network lasso (Hallac et al., 2015) alleviates this problem by using a different prediction model with a regularization term for each node

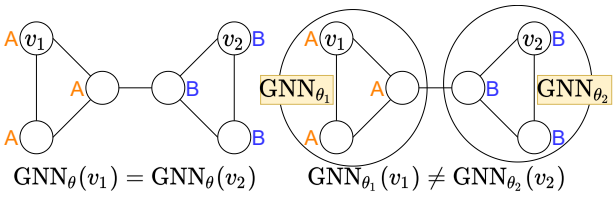

Figure 1: An example graph, where a GNN with parameter sharing is not able to distinguish nodes v_1 and v_2 that are isomorphic, although they belong to different classes. In this example, nodes belong to either class A or class B, and we assume node features to be the same across all nodes. Since v_1 and v_2 are symmetric, message-passing GNNs with parameter sharing will always generate the same predictions regardless of the depth of the model. In contrast, GraphAdaMix assigns nodes to multiple possibly overlapping clusters, and each node has its own embedding generator, which allows v_1 and v_2 to be distinguished. This figure is adapted from You et al. (2019).

in the graph. However, it is impractical for real-world graphs due to its huge time and space complexity.

Here we propose **Graph Ada**ptive **Mix**tures (GraphAdaMix), a technique that can be integrated with any graph representation learning models by relaxing the parameter-sharing restriction. Specifically, GraphAdaMix consists of K independent GNN models and a set of mixture parameters so that each node can select its most suitable model adaptively. As a result, nodes can be assigned to clusters with different patterns and node embeddings can be optimized simultaneously. For graph with semi-supervised settings, we develop an EM algorithm with mini-batch gradient descent to solve this problem in a scalable manner. The E-step approximates the posterior probability distribution of the latent cluster memberships, while the M-step updates the mixture parameters for downstream tasks. A proximity-driven regularization is additionally introduced into GraphAdaMix to learn mixtures for unlabeled nodes. In unsupervised tasks, GraphAdaMix is combined with the contrastive learning paradigm for model training. This enables the representations to be generalized to more different tasks. By jointly optimizing all GNN models and mixture parameters, GraphAdaMix effectively learns a discriminative embedding for each node. In addition, since GNN models used in the optimization process are agnostic to GraphAdaMix, the proposed framework can be well adapted to other graph-based models for performance enhancement.

This paper's main contributions can be summarized as follows:

- We propose a general and powerful framework, GraphAdaMix, to enhance the model capacities of representation learning on graphs. GraphAdaMix incorporates multiple independent GNNs with a mixture model for modeling different graph patterns. In general, GraphAdaMix can be combined with any graph representation learning algorithms.
- An EM algorithm and a contrastive learning paradigm is proposed to train the model in an end-to-end and scalable manner for semi-supervised and unsupervised tasks, respectively. By jointly updating the parameters of both GNNs and mixture models, GraphAdaMix is encouraged to learn more discriminative node embeddings.
- We demonstrate the effectiveness of GraphAdaMix through comprehensive experiments on ten real-world graph datasets. Experimental results show that GraphAdaMix consistently boosts various GNN variants in both semi-supervised and unsupervised settings.

2 RELATED WORKS

In general, GNN models learn embeddings by stacking multiple layers to aggregate information within a node's neighborhood. In many previous works, neighbors of the target node are equally treated in the aggregation process (Hamilton et al., 2017; Kipf and Welling, 2016a; Xu et al., 2019), while recent studies find it helpful to learn more powerful node embeddings by discriminating different nodes or special structures in the graph. For example, GAT (Veličković et al., 2017) proposes to learn attentive weights for different neighbors in the aggregation process, which helps to identify the important nodes from the irrelevant for the target. However, all these models apply the sharing scheme to every node in the graph. Thus, for any two isomorphic nodes (same local connectivity with the same neighborhoods' features), these models will always generate the same embeddings and therefore fail to distinguish the two nodes. In contrast, GraphAdaMix trains K independent GNN models with different parameters, and each node has its own mixture parameters, which increases the discriminative power of the model.

One way to further increase the model complexity of GNNs is to employ a multi-head attention mechanism (Veličković et al., 2017) by creating multiple independent GNN models. The node embedding can then be generated by averaging those representations (generated by each head). This allows the model to jointly aggregates different information or patterns from different GNN models. GraphAdaMix is similar to a

multi-head GAT in the sense that they both train K GNN models end-to-end with different parameters. However, in GraphAdaMix, instead of averaging across heads, we aggregate those K predictions from different GNN models by a weighted average, where the weights are different for different nodes. This allows the model to distinguish isomorphic nodes.

ClusterGCN (Chiang et al., 2019) separates the graph into multiple disjoint components by applying a graph clustering algorithm. A shared GCN can therefore be applied to each component, which scales typical GCNs to large graphs. On the contrary, GraphAdaMix clusters the nodes and trains the GNN models in an end-to-end fashion instead of precomputing the clusters deterministically. As a result, the clusters of nodes are adaptively chosen to minimize the loss function (e.g. the negative log-likelihood function). Also, GraphAdaMix relaxes the parameter sharing assumption and each node uses different GNN parameters according to its mixture distribution.

Instead of using the sharing scheme, Network Lasso (Hallac et al., 2015) assigns independent learnable weights for each node in a graph. The cost function of the model is specifically designed to be convex, which can be optimized through the Alternating Direction Method of Multipliers (ADMM) algorithm. Different from Network Lasso, GraphAdaMix allows the loss function to be non-convex, and it performs soft clustering instead of hard clustering so that each vertex can incorporate patterns from more than one cluster. Most importantly, since GraphAdaMix only requires updating the K set of GNN parameters while Network Lasso needs to update sets of model parameters with size equals to the number of nodes, GraphAdaMix can scale to large graphs with millions of nodes while Network LASSO can only handle graphs with thousands of nodes. Detailed comparisons between Network Lasso and GraphAdaMix can be found in the supplementary material.

3 PROBLEM DEFINITION

Let $\mathcal{G}$ be an undirected graph with vertex set $\mathcal{V} = \{v_i\}_{i=1}^N$ and edge set $\mathcal{E}$, i.e. $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each undirected edge $(i, j) \in \mathcal{E}$ is associated with a positive edge weight $w_{ij} \in \mathbb{R}_{>0}$ representing the degree of similarity between node i and node j. $\mathcal{N}(v)$ denotes the neighborhood of node v, i.e. $\mathcal{N}(v) = \{u \in \mathcal{V} \mid (v, u) \in \mathcal{E}\}$. We denote with $\mathbf{A} \in \mathbb{R}^{N \times N}$ the adjacency matrix and $\mathbf{X} \in \mathbb{R}^{N \times D_N}$ the node feature matrix. GraphAdaMix is model-agnostic. Therefore it can be applied to other types of graphs such as heterogeneous graphs, graphs with edge attributes, bipartite graphs, etc. For simplicity, we only consider homogenous graphs in this paper.

This paper focuses on the task of semi-supervised node classification and unsupervised node classification. Given the labels $\mathbf{Y}_L \in \{0, 1\}^{|\mathcal{V}_L| \times C}$ for a subset of nodes $\mathcal{V}_L \subset \mathcal{V}$ where C is the number of classes, the task is to infer the classes $\mathbf{Y}_U \in \{0, 1\}^{|\mathcal{V}_u| \times C}$ for the unlabeled nodes $\mathcal{V}_U = \mathcal{V} \setminus \mathcal{V}_L$ based on $\mathbf{X}$ and $\mathbf{A}$. For $n \in \mathcal{V}_L$, $\mathbf{y}_n \in \{0, 1\}^C$ and $[\mathbf{y}_n]_c = 1$ if node n belongs to class c. Node classifications on graphs are performed by first embedding nodes into a low-dimensional space followed by feeding them into a classifier.

4 GraphAdaMix

In this section, we introduce the GraphAdaMix architecture for transductive node classification. GraphAdaMix introduces K independent GNN models and a set of mixture parameters so that each node can adaptively select the most suitable model. It leverages the EM algorithm to achieve model clustering and optimization on graphs and consequently improves model capacity. Specifically, E-step approximates the posterior probability distribution of the latent cluster membership based on the K independent GNN models in the last iteration, and M-step updates the K GNN models and the mixture parameters based on the refined latent cluster membership for downstream tasks of node classification. The EM algorithm effectively maximizes the lower bound of the log-likelihood function.

4.1 Semi-supervised Node Classification with GraphAdaMix

In Graph Representation Learning with Adaptive Mixtures (GraphAdaMix), we consider K GNN models, each governed by its own parameter $\boldsymbol{\theta}_k$. Let $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K\}$ be the set of parameters of a GraphAdaMix model. We are interested in maximizing the likelihood (1) or the log-likelihood function (2) of the training data of the following form:

$$p(\mathbf{Y}_L \mid \mathbf{X}, \boldsymbol{\theta}) = \prod_{n \in \mathcal{V}_L} \sum_{k=1}^{K} \pi_{nk} \prod_{i=1}^{C} [\hat{\mathbf{y}}_n^{(k)}]_i^{[\mathbf{y}_n]_i} \tag{1}$$

$$\ln p(\mathbf{Y}_L \mid \mathbf{X}, \boldsymbol{\theta}) = \sum_{n \in \mathcal{V}_L} \ln \left(\sum_{k=1}^{K} \pi_{nk} \prod_{i=1}^{C} [\hat{\mathbf{y}}_n^{(k)}]_i^{[\mathbf{y}_n]_i} \right) \tag{2}$$

where $\pi_{nk}, n = 1, \ldots, N, k = 1, \ldots, K$ are the mixture parameters, and they must satisfy $0 \leq \pi_{nk} \leq 1$ and $\sum_{k=1}^{K} \pi_{nk} = 1$ for all $n \in \mathcal{V}$. $[\hat{\mathbf{y}}_n^{(k)}]_j$ is the prediction probability generated by the k-th GNN model on class j for node n. If $K = 1$, then $\pi_{nk} = 1$ for all $n \in \mathcal{V}$

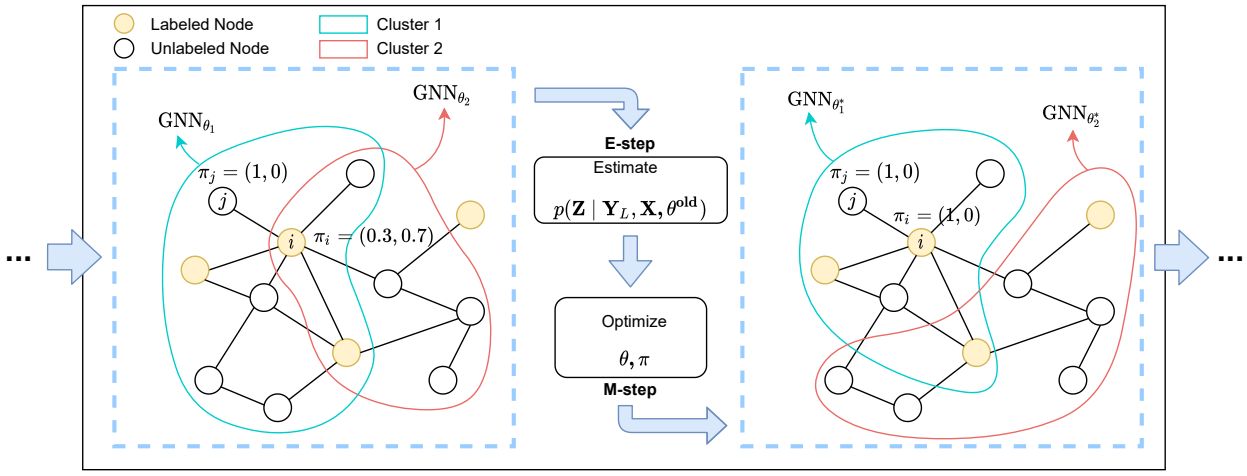

Figure 2: The semi-supervised training architecture of GraphAdaMix in an iterative training epoch. Nodes of the graph are assigned to different clusters based on the mixture distribution, where each cluster is associated with an independent GNN model. In each training epoch, the mixture assignments and GNN model parameters can be jointly updated by an EM algorithm. By iteratively optimizing the whole framework, GraphAdaMix learns cluster-specific GNN models for capturing different patterns of the graph.

and (1) and (2) become the likelihood and the log-likelihood of a vanilla GNN model.

It is difficult to optimize (2) due to the summation that occurs inside the logarithm. In order to maximize the log-likelihood function (2), we introduce a set of binary latent variables $\mathbf{Z} = \{\mathbf{z}_n, n \in \mathcal{V}_L\}$ where $\mathbf{z}_n \in \{0, 1\}^K$ is one-hot encoded, i.e. for each data point $n \in \mathcal{V}_L$,

$$z_{nk} = \begin{cases} 1 & \text{if the } k\text{-th GCN models node } n \\ 0 & \text{otherwise} \end{cases}$$

The complete-data likelihood and the complete-data log-likelihood can be written as:

$$p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}) = \prod_{n \in \mathcal{V}_L} \prod_{k=1}^{K} \left(\pi_{nk} \prod_{c=1}^{C} [\hat{\mathbf{y}}_n^{(k)}]_c^{[\mathbf{y}_n]_c} \right)^{z_{nk}} \tag{3}$$

$$\ln p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}) \tag{4}$$
$$= \sum_{n \in \mathcal{V}_L} \sum_{k=1}^{K} z_{nk} \left(\ln \pi_{nk} + \sum_{c=1}^{C} [\mathbf{y}_n]_c \ln [\hat{\mathbf{y}}_n^{(k)}]_c \right)$$

Let $q(\mathbf{Z})$ be some distribution over the latent variables $\mathbf{Z}$ satisfying $\sum_{\mathbf{Z}} q(\mathbf{Z}) = 1$, a lower bound of (2) can be derived as follows:

Proposition 1. *The log-likelihood* $\ln p(\mathbf{Y}_L \mid \mathbf{X}, \boldsymbol{\theta})$ *admits the following lower-bound*

$$\mathcal{L}(q, \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}) - \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln q(\mathbf{Z}) \tag{5}$$

where $q(\mathbf{Z})$ *is some distribution over the latent variables* $\mathbf{Z}$ *and the equality holds if and only if* $q(\mathbf{Z}) = p(\mathbf{Z} \mid \mathbf{Y}_L, \mathbf{X}, \boldsymbol{\theta})$

The proof of Proposition 1 can be found in the supplementary material.

4.1.1 EM Algorithm

The EM algorithm seeks to optimize the objective (1) by iteratively applying the E-step and the M-step. Suppose that the current parameters are $\boldsymbol{\theta}^{\text{old}}, \boldsymbol{\pi}^{\text{old}}$. In the E-step, we maximize the lower bound (5) with respect to $q(\mathbf{Z})$ while holding $\boldsymbol{\theta}^{\text{old}}, \boldsymbol{\pi}^{\text{old}}$ fixed. In the M-step, we maximize the lower bound (5) with respect to $\boldsymbol{\pi}, \boldsymbol{\theta}$ while holding $q(\mathbf{Z})$ fixed.

E-step : According to Prop. 1, the equality of (1) holds if and only if $q(\mathbf{Z}) = p(\mathbf{Z} \mid \mathbf{Y}_L, \mathbf{X}, \boldsymbol{\theta})$. Therefore, when holding $p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}^{\text{old}})$ fixed, maximizing the lower bound (5) with respect to $q(\mathbf{Z})$ amounts to finding:

$$\gamma_{nk} = p(z_{nk} = 1 \mid \mathbf{Y}_L, \mathbf{X}, \boldsymbol{\theta}) = \frac{\pi_{nk} \prod_{c=1}^{C} [\hat{\mathbf{y}}_n^{(k)}]_c^{[\mathbf{y}_n]_c}}{\sum_{i=1}^{K} \pi_{ni} \prod_{c=1}^{C} [\hat{\mathbf{y}}_n^{(i)}]_c^{[\mathbf{y}_n]_c}} \tag{6}$$

for all $n \in \mathcal{V}_L$ and $k \in \{1, 2, \ldots, K\}$.

M-step :

In M-step, we maximize the lower bound (5) with respect to $\boldsymbol{\pi}, \boldsymbol{\theta}$ by keeping $\boldsymbol{\gamma}$ fixed. The second term $-\sum_{\mathbf{Z}} q(\mathbf{Z}) \ln q(\mathbf{Z})$ can be treated as a constant since

it does not depend on $\boldsymbol{\pi}, \boldsymbol{\theta}$. Thus, we focus on minimizing $-\sum_{\mathbf{Z}} q(\mathbf{Z}) \ln p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta})$. By substituting Eq. 4 and 6, our optimization problem for M-step is given by

$$\min_{\boldsymbol{\pi},\boldsymbol{\theta}} - \sum_{n \in \mathcal{V}_L} \sum_{k=1}^{K} \gamma_{nk} \left(\ln \pi_{nk} + \sum_{c=1}^{C} [\mathbf{y}_n]_c \ln [\hat{\mathbf{y}}_n^{(k)}]_c \right) \quad (7)$$

We introduce the variables $\{\phi_{nk} \in \mathbb{R}, n = 1, \dots N, k = 1, \dots K\}$ and reparameterize π_{nk} using the softmax function as $\pi_{nk} = \frac{\exp \phi_{nk}}{\sum_{j=1}^{K} \exp \phi_{nj}}$. We optimize M-step objective function (Eq. 7) with respect to $\boldsymbol{\pi}$ and $\boldsymbol{\phi}$ by either batch gradient descent (for small graphs) or by mini-batch gradient descent (for large graphs).

To update $\boldsymbol{\theta}$ in the M-step, we perform one epoch training with gradient descent (for small graphs) or mini-batch gradient descent (for large graphs).

Please refer to the supplementary material for the entire algorithm.

4.1.2 Mixture Propagation

One drawback of using the objective function (7) is that for each non-training node j, the value of π_{jk} will not be updated. Therefore, we include an edge objective that penalizes the difference between the mixture distributions at adjacent nodes. The edge objective is implemented with $\frac{\rho}{|\mathcal{E}|} \sum_{(i,j) \in \mathcal{E}} \mathrm{JSD}(\boldsymbol{\pi}_i \parallel \boldsymbol{\pi}_j)$ where $\rho \geq 0$ is the trade-off hyperparameter and JSD is the Jensen–Shannon divergence.

Therefore, the optimization problem is given by

$$\min_{\pi,\boldsymbol{\theta}} \quad - \sum_{n \in \mathcal{V}_L} \sum_{k=1}^{K} \gamma_{nk} \left(\ln \pi_{nk} + \sum_{c=1}^{C} [\mathbf{y}_n]_c \ln [\hat{\mathbf{y}}_n^{(k)}]_c \right) + \frac{\rho}{|\mathcal{E}|} \sum_{(i,j) \in \mathcal{E}} \mathrm{JSD}(\boldsymbol{\pi}_i \parallel \boldsymbol{\pi}_j) \quad (8)$$

Note that in a connected graph, when $\rho \to +\infty$, all nodes will share the same mixture distribution.

4.2 Unsupervised node representation learning with GraphAdaMix

Mixture propagation (Sec. 4.1.2) assumes homophily (McPherson et al., 2001). In other words, we assume nodes that are closed to each other in the graph should apply a similar model mixture. However, this assumption may not hold in other datasets (Donnat et al., 2018; Henderson et al., 2012). More importantly, the learning of $\boldsymbol{\pi}$ (or $\boldsymbol{\phi}$) relies on the labeled data (Eq. 7). One may argue that when the training labels are limited, GraphAdaMix can easily overfit the training data as it has more degree of freedom (K times more parameters). To address both problems, we extend GraphAdaMix to unsupervised node representation learning, and in particular graph contrastive learning. Survey on graph contrastive learning can be found in Liu et al. (2021a,b). In graph contrastive learning, we generate multiple views for each node through stochastic graph augmentations. The views that are generated from the same nodes are called positive pairs, otherwise it is called negative pairs. The primary goal of contrastive learning is to maximize the agreement between positive pairs and minimize the agreement between negative pairs. Since the graph contrastive loss includes all nodes in the graph (unlike objective function 7 where it involves only the training nodes), applying graph contrastive learning to GraphAdaMix allows the mixture parameters π_{jk} to be updated for all nodes j.

In this paper, we focus our GraphAdaMix extension to the BGRL method (Thakoor et al., 2021) (due to its simplicity and its state-of-the-art performance). We adapt GraphAdaMix to BGRL by replacing its graph encoders with GraphAdaMix. Formally, in each epoch, BGRL-GraphAdaMix first generates two alternate views of the original graph: $\mathcal{G}_1$ and $\mathcal{G}_2$ by applying random node feature masking and random edge masking. Their node feature matrix and adjacency matrix are $(\tilde{\mathbf{X}}_1, \tilde{\mathbf{A}}_1)$ and $(\tilde{\mathbf{X}}_2, \tilde{\mathbf{A}}_2)$ respsectively. BGRL-GraphAdaMix maintains two graph encoders: the online encoder with parameter $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_K, \boldsymbol{\theta}_{\mathrm{M}}\}$ and the target encoder with parameter $\boldsymbol{\psi} = \{\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \dots, \boldsymbol{\psi}_K\}$. Let D_h be the embedding size and let $\tilde{\mathbf{H}}_1 \in \mathbb{R}^{N \times D_h}$ and $\tilde{\mathbf{H}}_2 \in \mathbb{R}^{N \times D_h}$ be the online representation and the target representation respectively where:

$$\tilde{\mathbf{H}}_1 := \sum_{k=1}^{K} \boldsymbol{\pi}_{(1,k)} \mathrm{GNN}_{\boldsymbol{\theta}_k}(\tilde{\mathbf{X}}_1, \tilde{\mathbf{A}}_1) \quad (9)$$

$$\tilde{\mathbf{H}}_2 := \sum_{k=1}^{K} \boldsymbol{\pi}_{(2,k)} \mathrm{GNN}_{\boldsymbol{\psi}_k}(\tilde{\mathbf{X}}_2, \tilde{\mathbf{A}}_2) \quad (10)$$

The target representation is fed into a multilayer perceptron (MLP) to generate a prediction of the target representation: $\tilde{\mathbf{Q}}_1 := \mathrm{MLP}_{\boldsymbol{\theta}_{\mathrm{M}}}(\tilde{\mathbf{H}}_1)$, where $\tilde{\mathbf{Q}}_1 \in \mathbb{R}^{N \times D_h}$.

Updating $\boldsymbol{\theta}$ and $\boldsymbol{\pi}_1$ We update $\boldsymbol{\theta}$ (and not $\boldsymbol{\psi}$) and the mixture parameters $\boldsymbol{\pi}_1$ by minimizing the cosine similarity between $\mathbf{q}_{1,i}$ and $\mathbf{h}_{2,i}$ for every node i, where $\mathbf{q}_{1,i}$ and $\mathbf{h}_{2,i}$ are the ith row of $\tilde{\mathbf{Q}}_1$ and $\tilde{\mathbf{H}}_2$ respectively. Formally, we minimize the following loss function with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\pi}_1$ using gradient descent:

$$\ell(\boldsymbol{\theta}, \boldsymbol{\pi}_1) = -\frac{2}{N} \sum_{i=0}^{N-1} \frac{\mathbf{q}_{1,i} \mathbf{h}_{2,i}^{\intercal}}{\parallel \mathbf{q}_{1,i} \parallel \parallel \mathbf{h}_{2,i} \parallel} \quad (11)$$

Updating $\boldsymbol{\psi}$ and $\boldsymbol{\pi}_2$ Similar to BGRL, the parameters of the target encoder $\boldsymbol{\psi} = \{\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \ldots, \boldsymbol{\psi}_K\}$ and its mixture parameters $\boldsymbol{\pi}_2$ are updated as an exponential moving average of the online parameters $\boldsymbol{\theta}$ and $\boldsymbol{\pi}_1$, respectively, i.e.,

$$\boldsymbol{\psi} \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta} \tag{12}$$

$$\boldsymbol{\pi}_2 \leftarrow \tau\boldsymbol{\pi}_1 + (1-\tau)\boldsymbol{\pi}_1 \tag{13}$$

where τ is the smoothing factor where higher τ discounts older $\boldsymbol{\theta}$ and $\boldsymbol{\pi}_1$ faster. Note that similar to BGRL and BYOL (Grill et al., 2020), BGRL-GraphAdaMix does not collapse to trivial solutions. Other training details can be found in the supplementary material.

4.3 Assumptions and Limitations

As mentioned in Sec. 4.2, the learning of $\boldsymbol{\pi}$ using mixture propagation assumes homophily (McPherson et al., 2001). Many methods (Klicpera et al., 2019; Perozzi et al., 2014) share this assumption and most common datasets adhere to this principle. However, for other datasets, structural role-based similarity may be more suitable for determining the mixture distribution, namely that nodes with similar local structural patterns should apply similar models (Donnat et al., 2018; Henderson et al., 2012; Ribeiro et al., 2017).

Also, the time and space complexity of GraphAdaMix are at least K times larger than its original model as it requires K forward and backward pass during training and the additional training of the mixture parameters $\boldsymbol{\phi}$. Empirically, we found that a large value of K usually leads to overfitting in real-world datasets. A straightforward way to improve the computational efficiency of GraphAdaMix is to use hard clustering instead of soft clustering, which means $\boldsymbol{\pi}$ become one-hot encoded. This makes the M-step K times faster as it requires only one forward and backward pass for each node.

5 DETAILED COMPARISONS WITH NETWORK LASSO

GraphAdaMix shares some similarities with the Network Lasso model (Hallac et al., 2015). Network Lasso considers the following general convex optimization problem:

$$\underset{\theta_1,\ldots,\theta_N}{\text{minimize}} \sum_{i\in\mathcal{V}} f_i(\theta_i) + \lambda \sum_{(j,k)\in\mathcal{E}} w_{jk} \left|\left|\theta_j - \theta_k\right|\right|_2 \tag{14}$$

where the optimization variables are $\theta_1, \theta_2, \ldots, \theta_N \in \mathbb{R}^d$. Here $f_i : \mathbb{R}^d \mapsto \mathbb{R} \cup \{\infty\}$ is the lost function for node i, and the second term encourages adjacent nodes to have the same model parameters (i.e. it encourages $\theta_i = \theta_j$ for edge $(i,j) \in \mathcal{E}$). Hallac et al. (2015, 2017) proposed to optimize (14) using the ADMM algorithm. It is important to note that for some small enough λ, the optimal solution of (14) breaks into clusters of nodes, with θ_i the same across all nodes in the cluster. From the modeling perspective, Network Lasso is similar to GraphAdaMix since they both cluster the vertices and fit a model to each cluster simultaneously. However, in GraphAdaMix, we allow f_i to be non-convex in θ_i for $i \in \mathcal{V}$, and therefore more sophisticated models such as GNNs can be applied. Besides, GraphAdaMix performs soft clustering instead of hard clustering so that each vertex can belong to more than one cluster. Also, GraphAdaMix uses an efficient EM-algorithm to cluster the vertices and update the GNN parameters to maximize the likelihood function while Network Lasso solves (14) using ADMM. Finally, GraphAdaMix only requires updating the K set of GNN parameters while Network Lasso needs to update the N sets of model parameters. In practice, we found that Network Lasso is impractical for real-world graphs due to its huge time and space complexity.

6 EXPERIMENTS

6.1 Datasets

For semi-supervised node classification, we evaluate our model on five graph benchmarks, including four citation networks: Cora, Citeseer, Pubmed, ogbn-arxiv) and a product co-purchase network ogbn-products. For unsupervised node representation learning, we use the WikiCS (Mernyei and Cangea, 2020), Amazon-Computers, Amazon-Photos, Coauthor-CS, Coauthor-Physics (Shchur et al., 2018). Details of the datasets can be found in the supplementary material.

6.2 Baselines

For semi-supervised node classification, we use 4 state-of-the-art semi-supervised graph representation learning models for comparison: GCN (Kipf and Welling, 2016a), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2017), GraphSAINT (Zeng et al., 2019) and SIGN (Rossi et al., 2020). We also included a multi-layer perceptron (MLP) baseline which does not use any structural information.

For unsupervised node representation learning, we compare BGRL-GraphAdaMix against 6 state-of-the-art baselines including DeepWalk (Perozzi et al., 2014), DGI (Velickovic et al., 2019), GMI (Peng et al., 2020), MVGRL (Hassani and Khasahmadi, 2020), GRACE (Zhu et al., 2020), and BGRL (Thakoor et al.,

2021).

Detailed descriptions of these baseline models can be found in the supplementary material.

6.3 Experiment Configuration

We tune the penalty parameter ρ and the number of independent GNN models K in GraphAdaMix using the validation set where ρ is selected from $\{0, 10, 30, 50, 70, 90, 100, 150, 200, 250, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000, 4000, 10000\}$ and K is selected form $\{2, 3, 4, 6, 8\}$. For Cora, Citeseer, and Pubmed, we follow the same experimental setup of (Kipf and Welling, 2016a) with the same train/validation/test split and the same hyperparameters. For ogbn-arxiv and ogbn-products, we use the same training, validation, and test set provided by Hu et al. (2020).

To evaluate the quality of the unsupervisely-learned embeddings, we follow the linear evaluation paradigm introduced by prior works (Thakoor et al., 2021; Velickovic et al., 2019). We first train each model in a fully unsupervised manner and generate the frozen embeddings for each node. These frozen embeddings are then fed into a simple logistic regression model, and it is trained without updating the graph neural networks. We follow the exact same configuration as in Thakoor et al. (2021) with the same hyperparameters (learning rate, embedding size, node feature masking, edge masking probabilities, etc.), and we select K from $\{2, 4, 8, 16\}$. Additional experimental details can be found in the supplementary material and Thakoor et al. (2021).

6.4 Results

For semi-supervised node classification, the results are summarized in Table 1, where we see that the use of adaptive mixtures consistently and significantly boosts the accuracy of a wide variety of state-of-the-art models across all of these baselines. The only exceptions are the Pubmed dataset and the Citeseer dataset where GraphAdaMix fails to improve the GCN model and the GAT model, resulting in a slight decrease in accuracy. It is also interesting to note that although the number of parameters is K times more than the baselines, overfitting does not occur.

For unsupervised node representation learning, we report all baselines' performances from published results. The results are summarized in Table 7 in the supplementary material. As we can see, BGRL-GraphAdaMix performs competitively and all datasets, and it outperforms all the baselines (including the supervised GCN baseline!) in all considered datasets.

7 ABLATION STUDIES

In this section, we seek to investigate the influence of the trade-off hyperparameters ρ and the number of independent GNN models K on the node classification accuracy.

7.1 Influence of ρ

We train the GCN-GraphAdaMix model on the Cora, Citeseer, Pubmed, ogbn-arxiv, and ogbn-products datasets. We select ρ from the range of $[0, 10000]$. As we can see in Fig. 3, for the Cora, Citeseer, and Pubmed datasets, we found that when ρ is small, the model overfits the dataset and performs badly on the testing set. We also observe similar results in ogbn-arxiv and ogbn-products. This validates the effectiveness of the mixture propagation mechanism (Sec. 4.1.2). For unsupervised node representation learning, we found that ρ does not influence much on the classification accuracy. The plots for BGRL-GraphAdaMix can be found in the supplementary material.

7.2 Influence of K

For semi-supervised node classification, we found that setting $k = 2$ is the optimal choice for the Cora, Citeseer, Pubmed datasets, ogbn-arxiv and ogbn-products datasets. Fig. 3 shows the results for the Cora, Citeseer and Pubmed datasets. In practice, one can tune the value of K by evaluating the performance on the validation set. In our experiments, we found that as we increase K, the mixture coefficient π_{nk} for some k will be close to zero for all $n \in \mathcal{V}$. Inspired by this, a heuristic is to start with a relatively large K, say 10, and then decide the values of K by observing the number of columns in the $\boldsymbol{\pi}$ matrix that are not all close to zero.

For unsupervised node representation learning, as indicated in Table 7 in the supplementary material, we found that there are positive correlation between K and the number of nodes in the graph. However, we argue that K depends more on the number of factors (or amount of information) hidden from the dataset rather than on the size of the dataset. Consider the following: One way to predict housing prices on a graph (neighboring houses (nodes) are connected by edges) is to employ GNN to perform regression (Hallac et al., 2015). However, spatial factors are usually unknown and difficult to quantify (e.g. distance to public transport, distance to schools, etc.), therefore similar houses (e.g. similar number of bathrooms,

Table 1: Average test accuracy comparisons for semi-supervised node classification.
OOM: Out-of-memory error.
*: We report the accuracy in the original paper where the standard error is not available

	Cora	Citeseer	Pubmed	ogbn-arxiv	ogbn-products
MLP	55.1*	46.5*	71.4*	55.50 ± 0.0023	61.06 ± 0.0008
MLP-GraphAdaMix	$\mathbf{57.17 \pm 0.88}$	$\mathbf{54.70 \pm 1.30}$	$\mathbf{71.82 \pm 1.16}$	$\mathbf{57.19 \pm 0.52}$	$\mathbf{61.92 \pm 0.0021}$
GCN	**81.5***	70.3*	79.0*	71.74 ± 0.0029	OOM
GCN-GraphAdaMix	81.13 ± 0.73	$\mathbf{71.78 \pm 0.55}$	$\mathbf{81.00 \pm 0.61}$	$\mathbf{71.75 \pm 0.0033}$	OOM
GraphSAGE	80.02 ± 0.83	69.91 ± 0.99	77.13 ± 0.64	71.49 ± 0.0027	78.29 ± 0.0016
GraphSAGE-GraphAdaMix	$\mathbf{80.25 \pm 0.55}$	$\mathbf{70.67 \pm 0.64}$	$\mathbf{77.89 \pm 0.32}$	$\mathbf{71.80 \pm 0.026}$	$\mathbf{79.38 \pm 0.0028}$
GAT	83.00 ± 0.51	$\mathbf{72.50 \pm 0.7}$	79.00 ± 0.3	OOM	OOM
GAT-GraphAdaMix	$\mathbf{83.02 \pm 0.44}$	71.46 ± 0.37	$\mathbf{79.73 \pm 0.66}$	OOM	OOM
GraphSAINT	78.23 ± 2.08	63.51 ± 4.99	75.03 ± 0.46	58.34 ± 0.16	79.08 ± 0.0024
GraphSAINT-GraphAdaMix	$\mathbf{80.54 \pm 0.60}$	$\mathbf{67.58 \pm 1.7}$	$\mathbf{75.34 \pm 0.36}$	$\mathbf{67.24 \pm 1.70}$	$\mathbf{79.12 \pm 0.010}$
SIGN	74.21 ± 2.15	64.19 ± 1.82	75.53 ± 1.13	70.83 ± 0.10	74.42 ± 0.021
SIGN-GraphAdaMix	$\mathbf{77.94 \pm 1.17}$	$\mathbf{67.17 \pm 1.55}$	$\mathbf{75.68 \pm 0.78}$	$\mathbf{70.98 \pm 0.14}$	$\mathbf{75.33 \pm 0.33}$

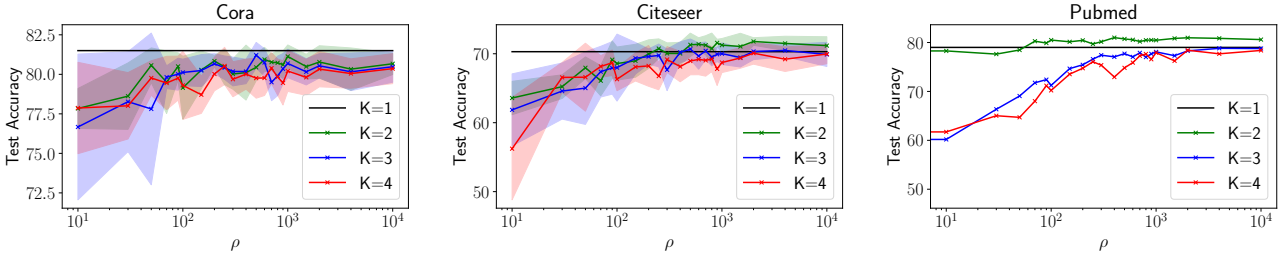

Figure 3: Influence of K and ρ on node classification accuracy using GCN-GraphAdaMix. Markers denote mean testing classification accuracy over 10 runs. Shaded areas denote standard error. K is selected from $\{1, 2, 3, 4\}$ and ρ is selected from $\{0, 10, 30, 50, 70, 90, 100, 150, 200, 250, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000, 4000, 10000\}$. **Best viewed in color.**

square footage, etc.) can have drastically different prices. GraphAdaMix allows different house groups to share different GNN models, without having to use the potentially misleading pricing model from other locations. The plots for BGRL-GraphAdaMix can be found in the supplementary material.

8 QUALITATIVE ANALYSIS

We performed qualitative analysis on the node representations learned by the GraphAdaMix model to better understand the properties of GraphAdaMix. We visualize the embeddings learned by the K GNN models using t-SNE projection (Van der Maaten and Hinton, 2008). The plots are given in Figure 4. As we can see, the clusters of the learned embeddings are clearly defined, which implies same GNN models produce similar embeddings and different GNN models produce different embeddings. In other words, those K independent GNN captures different patterns in the graph by adaptively choosing the mixture models.

We also perform further analyses and uses t-SNE to project the concatenation of the online representations and the target representation (i.e. $[\tilde{\mathbf{H}}_1 \parallel \tilde{\mathbf{H}}_2]$) into the 2D space. The scatter plots are given in Figure 5. As we can see, the learned embeddings' t-SNE 2D projection generates a distinguishable clustering in the 2D space comparing to the raw features and the Randomly-initialized BGRL-GraphAdaMix model, obtaining a Silhouette score (Rousseeuw, 1987) of 0.265, meaning that BGRL-GraphAdaMix can capture class-specific information even when it is trained without using node labels.

9 CONCLUSION

In this paper, we propose Graph Representation Learning with Adaptive Mixtures (GraphAdaMix). It

Table 2: Average test accuracy comparisons for unsupervised node representation learning.

	WikiCS	Am. Computers	Am.Photos	CoauthorCS	CoauthorPhy
Raw features	71.98 ± 0.00	73.81 ± 0.00	78.53 ± 0.00	90.37 ± 0.00	93.58 ± 0.00
DeepWalk	74.35 ± 0.06	85.68 ± 0.06	89.44 ± 0.11	84.61 ± 0.22	91.77 ± 0.15
DeepWalk+Features	77.21 ± 0.03	86.28 ± 0.07	90.05 ± 0.08	87.70 ± 0.04	94.90 ± 0.09
DGI	75.35 ± 0.14	83.95 ± 0.47	91.61 ± 0.22	92.15 ± 0.63	94.51 ± 0.52
GMI	74.85 ± 0.08	82.21 ± 0.31	90.68 ± 0.17	OOM	OOM
MVGRL	77.52 ± 0.08	87.52 ± 0.11	91.74 ± 0.07	92.11 ± 0.12	95.33 ± 0.03
GRACE	78.19 ± 0.01	87.46 ± 0.22	92.15 ± 0.24	92.93 ± 0.01	95.26 ± 0.02
Random-Init	78.95 ± 0.58	86.46 ± 0.38	92.08 ± 0.48	91.64 ± 0.29	93.71 ± 0.29
GRACE	80.14 ± 0.48	89.53 ± 0.35	92.78 ± 0.45	91.12 ± 0.20	OOM
BGRL	79.36 ± 0.53	89.68 ± 0.31	92.87 ± 0.27	93.21 ± 0.18	95.56 ± 0.41
BGRL-GraphAdaMix ($K = 2$)	**81.72 ± 0.01**	**89.73 ± 1.15**	**94.71 ± 0.60**	93.31 ± 0.29	95.95 ± 0.32
BGRL-GraphAdaMix ($K = 4$)	81.57 ± 0.01	88.83 ± 0.47	93.99 ± 0.64	93.33 ± 0.96	95.81 ± 0.23
BGRL-GraphAdaMix ($K = 8$)	77.80 ± 0.57	87.67 ± 0.55	94.10 ± 0.56	**93.88 ± 0.48**	96.03 ± 0.29
BGRL-GraphAdaMix ($K = 16$)	77.64 ± 1.09	85.48 ± 0.68	93.20 ± 1.15	93.63 ±0.25	**96.61 ± 0.30**
Supervised GCN	*77.19 ± 0.12*	*86.51 ± 0.54*	*92.42 ± 0.22*	*93.03 ± 0.31*	*95.65 ± 0.16*

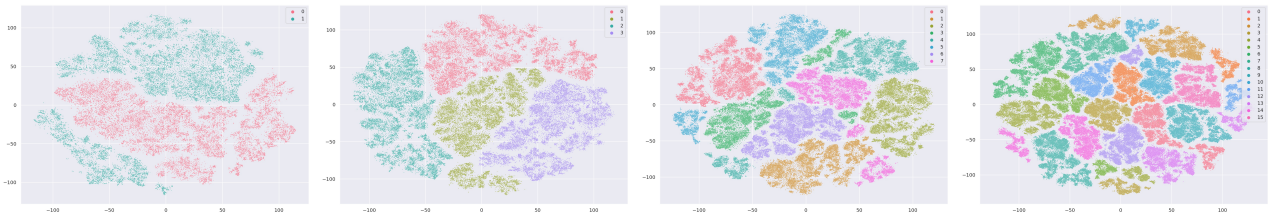

Figure 4: t-SNE projections of the node embeddings in the Coauthor-Physics dataset from the features extracted from a learned BGRL-GraphAdaMix model with K equals to $2, 4, 8, 16$ from left to right. Each scatter plot has $N * K$ dots as GraphAdaMix generates K different embeddings for each node before the mixing. A dot in the scatter plot corresponds to a node embedding generated by one of the K independent GNN. The color indicates which GNN generates the embedding. The embeddings generated by the same GNN tends to form a cluster and these clusters are clearly separated. **Best viewed in color.**

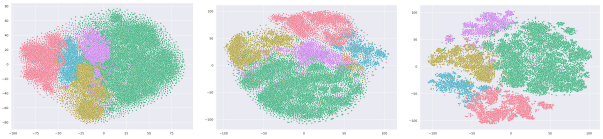

Figure 5: t-SNE projections of the nodes in the Coauthor-Physics dataset from the raw features (left), features from a randomly initialized BGRL-GraphAdaMix model (middle), and a learned BGRL-GraphAdaMix model (right). The colors indicate the node classes. The Silhouette score from left to right is 0.153, 0.219, 0.265. The clusters of the learned BGRL-GraphAdaMix model's embeddings are clearly identified. **Best viewed in color.**

can be readily adapted to many graph-based representation learning models for performance enhancement. Extensive experiments on semi-supervised node classification and unsupervised node representation learning across different GNN backbones show that GraphAdaMix consistently improves the performance of a wide range of models. There are many applications and extensions of GraphAdaMix that remain to be explored.

References

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. (2019). Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD*, pages 257–266.

Donnat, C., Zitnik, M., Hallac, D., and Leskovec, J. (2018). Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD*, pages 1320–1329.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D. (2019). Graph neural networks for social recommendation. In *WWW*, pages 417–426.

Fey, M. and Lenssen, J. E. (2019). Fast graph rep-

resentation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., Piot, B., kavukcuoglu, k., Munos, R., and Valko, M. (2020). Bootstrap your own latent - a new approach to self-supervised learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21271–21284. Curran Associates, Inc.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings.

Hallac, D., Leskovec, J., and Boyd, S. (2015). Network lasso: Clustering and optimization in large graphs. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 387–396.

Hallac, D., Wong, C., Diamond, S., Sharang, A., Sosic, R., Boyd, S., and Leskovec, J. (2017). Snapvx: A network-based convex optimization solver. *The Journal of Machine Learning Research*, 18(1):110–114.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034.

Hassani, K. and Khasahmadi, A. H. (2020). Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pages 4116–4126. PMLR.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C., and Li, L. (2012). Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

Kapoor, A., Ben, X., Liu, L., Perozzi, B., Barnes, M., Blais, M., and O'Banion, S. (2020). Examining covid-19 forecasting using spatio-temporal graph neural networks. *arXiv preprint arXiv:2007.03113*.

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

Kipf, T. N. and Welling, M. (2016b). Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*.

Klicpera, J., Weißenberger, S., and Günnemann, S. (2019). Diffusion improves graph learning. In *NIPS*, pages 13354–13366.

Liu, X., Zhang, F., Hou, Z., Mian, L., Wang, Z., Zhang, J., and Tang, J. (2021a). Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*.

Liu, Y., Pan, S., Jin, M., Zhou, C., Xia, F., and Yu, P. S. (2021b). Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111*.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

McPherson, M., Smith-Lovin, L., and Cook, J. M. (2001). Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444.

Mernyei, P. and Cangea, C. (2020). Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *NIPS*, pages 8024–8035.

Peng, Z., Huang, W., Luo, M., Zheng, Q., Rong, Y., Xu, T., and Huang, J. (2020). Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference 2020*, pages 259–270.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD*, pages 701–710.

Ribeiro, L. F., Saverese, P. H., and Figueiredo, D. R. (2017). struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD*, pages 385–394.

Rossi, E., Frasca, F., Chamberlain, B., Eynard, D., Bronstein, M., and Monti, F. (2020). Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. (2018). Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*.

Sinha, A., Shen, Z., Song, Y., Ma, H., Eide, D., Hsu, B.-J., and Wang, K. (2015). An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pages 243–246.

Thakoor, S., Tallec, C., Azar, M. G., Munos, R., Veličković, P., and Valko, M. (2021). Bootstrapped representation learning on graphs. In *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. (2019). Deep graph infomax. *ICLR (Poster)*, 2(3):4.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *ICLR*.

You, J., Ying, R., and Leskovec, J. (2019). Position-aware graph neural networks. In *International Conference on Machine Learning*, pages 7134–7143. PMLR.

Yu, B., Yin, H., and Zhu, Z. (2017). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. (2019). Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*.

Zhu, Y., Xu, Y., Liu, Q., and Wu, S. (2021). An Empirical Study of Graph Contrastive Learning. *arXiv.org*.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. (2020). Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*.

Supplementary Material: GraphAdaMix: Enhancing Node Representations with Graph Adaptive Mixtures

A PROOF OF PROPOSITION 1

In this section, we present the detailed proof of Proposition 1

We will show that the log-likelihood admits the following lower-bound

$$\sum_{\mathbf{Z}} q(\mathbf{Z}) \ln p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}) - \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln q(\mathbf{Z})$$

and equality holds if and only if $q(\mathbf{Z}) = p(\mathbf{Z} \mid \mathbf{Y}_L, \mathbf{X}, \boldsymbol{\theta})$

Proof.

$$\ln p(\mathbf{Y}_L \mid \mathbf{X}, \boldsymbol{\theta}) = \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln p(\mathbf{Y}_L \mid \mathbf{X}, \boldsymbol{\theta}) \tag{15}$$

$$= \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \frac{p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta})}{p(\mathbf{Z} \mid \mathbf{Y}_L, \mathbf{X}, \boldsymbol{\theta})} \tag{16}$$

$$= \sum_{\mathbf{Z}} q(\mathbf{Z}) \left[\ln \frac{p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} + \ln \frac{q(\mathbf{Z})}{p(\mathbf{Z} \mid \mathbf{Y}_L, \mathbf{X}, \boldsymbol{\theta})} \right] \tag{17}$$

$$= \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \frac{p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} + \mathrm{KL}(q \parallel p) \tag{18}$$

$$\geq \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \frac{p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} \tag{19}$$

$$\begin{aligned} &= \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln p(\mathbf{Y}_L, \mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}) \\ &\quad - \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln q(\mathbf{Z}) \end{aligned} \tag{20}$$

$$= \mathcal{L}(q, \boldsymbol{\theta}, \boldsymbol{\pi}) \tag{21}$$

where (19) uses the fact that $\mathrm{KL}(q \parallel p) \geq 0$. Also, the equality hold if and only if $q(\mathbf{Z}) = p(\mathbf{Z} \mid \mathbf{Y}_L, \mathbf{X}, \boldsymbol{\theta})$. In the remaining sections, we will focus on maximizing the lower-bound $\mathcal{L}(q, \boldsymbol{\theta}, \boldsymbol{\pi})$. □

B SEMI-SUPERVISED NODE CLASSIFICATION WITH GraphAdaMix

The training procedure of GraphAdaMix for semi-supervised node classification

C DETAILS OF THE DATASETS

C.1 Semi-supervised node classification with GraphAdaMix

For semi-supervised node classification, we evaluate our model on five graph benchmarks, including four citation networks: Cora, Citeseer, Pubmed, ogbn-arxiv, and a product co-purchase network ogbn-products.

Algorithm 1 GraphAdaMix

Input: K untrained GNN model with parameter $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \dots \boldsymbol{\theta}_K\}$, an observed graph $\mathcal{G}$ with node features $\mathbf{X}$, and training labels $\mathbf{Y}_L$

Output: Predicted labels for the unlabeled nodes $\mathcal{V}_U$

Randomly initialize $\boldsymbol{\theta}$
Initialize $\boldsymbol{\phi}$ as a zero matrix
for iteration $t = 1, \dots, T$ **do**
▷ E-Step
  for $n \in \mathcal{V}_L$ **do**
    for $k = 1 \dots K$ **do**
      Set $\gamma_{nk} = p(z_{nk} = 1 \mid \mathbf{Y}_L, \mathbf{X}, \boldsymbol{\theta})$ according to Equation 6.
▷ M-Step
  Update $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ to optimize the objective function (8) with SGD by setting $\boldsymbol{\gamma}$ fixed.
Predict categorical distribution for the unlabeled vertex $n \in \mathcal{V}_U$ by $p([\mathbf{y}_n]_j = 1 \mid \mathbf{X}, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_{nk} [\hat{\mathbf{y}}_n^{(k)}]_j$
return Predicted labels: $\{\text{argmax}_j \, p([\mathbf{y}_n]_j = 1 \mid \mathbf{X}, \boldsymbol{\theta})\}_{n \in \mathcal{V}_U}$

Each dataset forms an undirected and unweighted graph. Detailed descriptions are listed as follows:

Cora: This dataset contains 2708 papers from 7 classes: Case-Based, Genetic Algorithm, Neural Networks, Probabilistic Models, Reinforcement Learning, Rule Learning, Theory. Each paper indicates a node in the graph and an undirected edge is assigned to a pair pair nodes if one is cited by the other. Each node is represented by a 1433-d one-hot vector, where each element indicates the presence/ absence of a specific word in the corresponding paper.

Citeseer: Similar to the Cora dataset, Citeseer includes a set of papers, each of which is collected from one of the following classes: Agents, ArtificialIntelligence, Database, Human-Computer Interaction, Information Retrieval, and Machine Learning. The definitions of the graph and raw attributes of each node are the same as Cora.

Pubmed: The setting of this dataset is similar to Cora, except that all papers are collected from three classes on medical science (i.e., Diabetes Mellitus Experimental, Diabetes Mellitus Type 1, and Diabetes Mellitus Type 2) and each paper/ node is represented by a TF/ITF weighted word vector.

ogbn-arxiv (Hu et al., 2020): This dataset contains $169,343$ arXiv papers collected from 40 classes. Each paper/ node is represented by a 128-d feature vector, which is obtained by averaging the word embedding of its titles and abstract.

ogbn-products (Hu et al., 2020): This dataset is an undirected graph representing a product co-purchase network of Amazon. Each node is a product and an edge between two nodes indicates that the two products are purchased together. Node attribute vectors are generated from bag-of-words features followed by a 100-d PCA operation.

C.2 Unsupervised node representation learning with BGRL-GraphAdaMix

For unsupervised node representation learning, we use the WikiCS (Mernyei and Cangea, 2020), Amazon-Computers, Amazon-Photos, Coauthor-CS, Coauthor-Physics (Shchur et al., 2018).

WikiCS (Mernyei and Cangea, 2020): This dataset contains $11,701$ articles collected from Wikipedia. Each article/ node is represented by a 300-d feature vector, which is obtained by averaging the GloVE embeddings (Pennington et al., 2014) of all words in the article. Each article belongs to one of the 10 classes based on its subfield.

Amazon-Computers, Amazon-Photos: Similar to ogbn-products, these two graphs are undirected graphs representing a product co-purchase network of Amazon where each node is a product and an edge indicates that the two products are frequently purchased together. Each product belongs to one of the 10 (for Amazon-Computers) and 8 (for Amazon-Photos) classes based on the product category. Node attribute vectors are generated from bag-of-words features of a product's reviews

Coauthor-CS, Coauthor-Physics These two graphs are undirected co-authorship graphs. These graphs are collected from the Microsoft Academic Graph Sinha et al. (2015). Each author/ node is represented by a $6,805$

(for Coauthor-CS) and a $8,415$ (for Coauthor-Physics) dimensional feature vector, which is generated from bag-of-words features of the keywords of the author's papers. Each author belongs to one of the 15 (for Coauthor-CS) and 5 (for Coauthor-Physics) classes based on the author's research field.

D EXPERIMENTAL DETAILS

Table 3: Dataset statistics and hyperparameters for semi-supervised node classification with GraphAdaMix

	Cora	Citeseer	Pubmed	ogbn-arxiv	ogbn-products
# Nodes	2,708	3,327	19,717	169,343	2,449,029
# Edges	5,429	4,732	44,338	1,166,243	61,859,140
# Features	1,433	3,703	500	128	100
# Classes	7	6	3	40	47
# Training	140	120	60	90941	196,615
# Valid	500	500	500	29,799	39,323
# Test	1,000	1,000	1,000	48,603	2,213,091
# Epochs	200	200	200	200	300
Learning rate	0.05	0.05	0.05	0.01	0.01
# Layers	2	2	2	3	3
Hidden sizes	16	16	16	256	256
Embedding sizes	16	16	16	256	256
Dropout rate	0.5	0.5	0.5	0.5	0.5

Table 4: Dataset statistics and hyperparameters for unsupervised node representation learning with BGRL-GraphAdaMix

	WikiCS	Amazon Computers	Amazon Photos	Coauthor CS	Coauthor Physics
# Nodes	11,701	13,752	7,650	18,333	34,493
# Edges	216,123	245,861	119,081	81,894	247,962
# Features	300	767	745	6,805	8,415
# Classes	10	10	8	15	5
# Training	10%	10%	10%	10%	10%
# Valid	10%	10%	10%	10%	10%
# Test	80%	80%	80%	80%	80%
$p_{f,1}$	0.2	0.2	0.1	0.3	0.1
$p_{f,2}$	0.1	0.1	0.2	0.4	0.4
$p_{e,1}$	0.2	0.5	0.4	0.3	0.4
$p_{e,2}$	0.3	0.4	0.1	0.2	0.1
η_{base}	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	10^{-4}	10^{-5}	10^{-5}
embedding size	256	128	256	256	128
$\mathcal{E}$ hidden sizes	512	256	512	512	256
$\text{MLP}_{\boldsymbol{\theta}_{\text{M}}}$ hidden sizes	512	512	512	512	512
# layers	2	2	2	2	2

D.1 Baselines

D.1.1 Semi-supervised node classification with GraphAdaMix

MLP: We implement a multi-layer perceptron (MLP) to be one of the baselines, which shows the performance of learning without incorporating structural information.

GCN: GCN learns node embeddings by stacking multiple graph convolutional layers where techniques such as mini-batch training and neighborhood sampling are not used.

GraphSAGE: This method applies spatial aggregation for each GNN layer. Different from GCN, GraphSAGE can be trained with mini-batch gradient descent, where node-level neighbor-sampling can be applied in the aggregation process to improve the scalability of the model.

GraphSAINT: GraphSAINT is a general framework proposed for training in large-scale graphs with a mini-batch setting. Different from GraphSAGE, GraphSAINT applies a graph-level sampling strategy with variance reduction techniques in the training process.

SIGN: This method pre-computes all the diffused features of a node within multi-hop neighbors. The model can therefore learn from the pre-computed diffused features without message-passing during the training process, which makes it feasible to be trained in large-scale graphs.

D.1.2 Unsupervised node representation learning with BGRL-GraphAdaMix

For unsupervised node representation learning, we compare BGRL-GraphAdaMix with the following baselines: DeepWalk (Perozzi et al., 2014), DGI (Velickovic et al., 2019), GMI (Peng et al., 2020), MVGRL (Hassani and Khasahmadi, 2020), GRACE (Zhu et al., 2020), BGRL (Thakoor et al., 2021), and a supervised-learning baseline.

Notice that most of them are graph contrastive learning methods (expect the supervised-learning baseline). In general, Graph Contrastive Learning (GCL) trains two models, a node encoder and a discriminator simultaneously. The node encoder generates node representations and the discriminator distinguishes semantically similar representation pairs from the dissimilar pairs. The main difference among previous works lies in how they define the notion of positive and negative pairs.

The descriptions of the baselines for unsupervised representation learning are listed as follows:

Raw features: We simply treat the raw node features as the frozen embeddings and fed it into a logistic regression model.

DeepWalk: DeepWalk (Perozzi et al., 2014) is an unsupervised node representation method which follows the contrastive training paradigm. Specifically, it samples node sequences by random walk. Node-pairs that are co-occured in a random walk are treated as positive pairs and otherwise are treated as negative pairs. This exploits the homophily properties in many real world graphs.

DGI: Deep Graph Infomax (DGI) (Velickovic et al., 2019) is an unsupervised node representation learning method. It relies on maximizing mutual information between the node (local) representations and graph (global) representations. Unlike DeepWalk, it does not rely on random walk objectives and thus does not focus on preserving homophily information in the original graph. It also follows the contrastive training paradigm where node and graph representations computed from the original graph are considered as positive pairs. Random node shuffling is applied to the original graph to generate a corrupted graph. The node representations computed from the corrupted graph and the graph representations computed from the original graph are considered as negative pairs.

GMI: Graphical Mutual Information (Peng et al., 2020) is a generalization of the conventional mutual information computations from vector space to graph domain. Similar to DGI, GMI also relies on mutual information maximization. However, unlike DGI, GMI accounts for both the mutual information between features and the mutual information between edges, which is more fine-grained.

MVGRL: MVGRL (Hassani and Khasahmadi, 2020) is also a contrastive-learning based model. It maximizes the mutual information between representations encoded from different structural views of graphs. MVGRL uses graph diffusion and random node sampling to generate augmented graphs. Node representations and graph representations computed from augmented graphs are considered as positive examples. Similar to DGI, MVGRL also relies on negative samples. Towards this end, MVGRL generates corrupted graphs by applying random node shuffling. Node representations computed from the corrupted graph and the graph representations computed from the original or augmented graph are considered as negative pairs.

GRACE: GRACE (Zhu et al., 2020) also follows the contrastive learning paradigm. Similar to BGRL, they also use node features masking and random edges removal to generate augmented views of a graph. The embeddings of the same node in two augmented graphs are considered as positive pairs and otherwise (the inter-view negative pairs and the intra-view negative pairs) are considered as negative pairs. They use the InfoNCE (Gutmann and Hyvärinen, 2010) objective to maximize the mutual information between node representations in augmented

views.

Random-Init*: We also report the performance of Random-Init from DGI (Velickovic et al., 2019), where a randomly initialized encoder (with the same architecture as DGI) is used to generate the node embeddings.

BGRL: BGRL (Thakoor et al., 2021) is also a self-supervised learning that is based on contrastive learning. Inspired by BYOL (Grill et al., 2020), BGRL eliminates the need of negative samples by bootstrapping the output of a delayed version of its encoder. For positive examples, BGRL uses node features masking and random edges removal to generate augmented views of a graph. The embeddings of the same node in the two augmented graphs are considered as positive examples.

D.2 Training Details

All experiments are conducted on a single Linux machine with an AMD EPYC 7713 64-Core Processor, 630 GB RAM, and three A100-SXM4-40GB Graphics Card.

D.2.1 Semi-supervised node classification with GraphAdaMix

For semi-supervised node classification, we implement our method in PyTorch (Paszke et al., 2019) with the PyTorch Geometric Library (Fey and Lenssen, 2019). For all baselines, we use the PyTorch Geometric (Fey and Lenssen, 2019) implementations, and we closely follow the same procedure as Kipf and Welling (2016a); Veličković et al. (2017) to split the vertex set into the training set, validation set, and test set. For a fair comparison, we evaluate all methods under the same GNN backbone and the same hyperparameters.

For a fair comparison, we evaluate all methods under the same GNN backbone and the same hyperparameters. To tune the penalty parameter ρ and the number of independent GNN models K in GraphAdaMix, we run the multiple experiments, each with different ρ and K and the value that achieves the best accuracy on the validation set will be selected. In all datasets, ρ is selected from $\{0, 10, 20, 30, 40, 50, 80, 100, 300\}$ and K is selected form $\{2, 4, 6, 8\}$. In each experiment, we run for 200 epochs For **Cora, Citeseer, Pubmed**, we follow the same experimental setup of Kipf and Welling (2016a). Specifically, we use two-layers GNNs of 16 hidden features with a 0.5 dropout rate in all GNN layers. Finally, the optimal solution that achieves the best on the validation set are applied to the test set and the test performance are reported in the paper.

For **ogbn-arxiv** and **ogbn-products**, we use the same training, validation, and test set provided by Hu et al. (2020). For all baselines, we use a 3 layers model with 256 hidden features with a learning rate of 0.01.

Table 3 describes hyperparameters for most of our setups with GraphAdaMix.

D.2.2 Unsupervised node representation learning with BGRL-GraphAdaMix

For unsupervised node representation learning, we implement BGRL-GraphAdaMix using the GCL Library (Zhu et al., 2021). For unsupervised node representation learning, we follow the same training procedure and hyperparameter settings as BGRL (Thakoor et al., 2021).

Specifically, in all our experiments, we run BGRL-GraphAdaMix, for 10,000 epochs on all datasets. The BGRL-GraphAdaMix predictor $\text{MLP}_{\boldsymbol{\theta}_\text{M}}$ is implemented with one hidden layer. For all dataset, in each layer including the final layer, we apply first the batch normalization (Ioffe and Szegedy, 2015) with decay rate 0.99, and then the PReLU (He et al., 2015) activation in all datasets. The random node feature masking probability (p_{f_1} for the first view and p_{f_2} for the second view) and the random edge masking probability (p_{e_1} for the first view and p_{e_2} for the second view) are shown in Table 4. Finally, in all datasets, we also follow BGRL to use Glorot initialization (Glorot and Bengio, 2010) to initialize all model parameters. AdamW optimizer (Loshchilov and Hutter, 2017) with 10^{-5} weight decay is used to optimize the model parameters (including the mixture parameters $\boldsymbol{\pi}$). The settings of the base learning rate η_base can be found in Table. 4. We also adjust the learning rate using a cosine schedule in the training phase where the details can be found in Thakoor et al. (2021).

D.3 Training Complexity

Figure 5 shows the training time for GCN and BGRL model with different values of K. Since the optimal value of K usually falls within a narrow range of $K \in \{1, 2, 3, 4\}$, the incrase in training time is limited, and this

Table 5: Further comparison of training time (in seconds)

	Number of epochs	Cora	Citeseer	Pubmed
GCN	1,000	9	9	18
GCN-GraphAdaMix (K=2)	1,000	14	14	24
GCN-GraphAdaMix (K=3)	1,000	20	20	29
GCN-GraphAdaMix (K=4)	1,000	27	24	34
BGRL	10,000	172	227	743
BGRL-GraphAdaMix (K=2)	10,000	258	290	860
BGRL-GraphAdaMix (K=3)	10,000	373	368	1,005
BGRL-GraphAdaMix (K=4)	10,000	483	487	1,092

approach stands from a practical perspective. Assuming the number of features is fixed for all layers, the time complexity of GCN and GCN-GraphAdam are $O(L\|A\|_0F + LNF^2)$ and $O(L\|A\|_0FK + LNF^2K)$ respectively, where L is the number of layers, $\|A\|_0$ is the number of nonzeros in the adjacency matrix, and F is the number of features. The memory complexity of GCN and GCN-GraphAdam are $O(LNF + LF^2)$ and $O(LNFK + LF^2K)$ respectively.

D.4 Parameter-efficiency

Table 6: Average test accuracy comparisons for GCN with increase number of parameters.

	Cora (number of parameters)	Citeseer (number of parameters)	Pubmed (number of parameters)
GCN (hidden size = 32)	$\mathbf{81.31 \pm 0.47}$ $(\mathbf{46,119})$	70.78 ± 0.55 $(118,726)$	79.29 ± 0.41 $(16,131)$
GCN-GraphAdaMix ($K = 2$)	81.13 ± 0.73 $(46,126)$	$\mathbf{71.50 \pm 1.04}$ $(\mathbf{118,732})$	$\mathbf{81.00 \pm 0.61}$ $(\mathbf{16,134})$
GCN (hidden size = 48)	$\mathbf{81.46 \pm 0.75}$ $(\mathbf{69,175})$	$\mathbf{70.60 \pm 0.26}$ $(\mathbf{178,086})$	$\mathbf{79.07 \pm 0.64}$ $(\mathbf{24,195})$
GCN-GraphAdaMix ($K = 3$)	81.21 ± 0.81 $(69,189)$	70.48 ± 0.88 $(178,098)$	78.81 ± 0.49 $(24,201)$
GCN (hidden size = 64)	$\mathbf{81.13 \pm 0.72}$ $(\mathbf{92,231})$	$\mathbf{70.35 \pm 0.76}$ $(\mathbf{237,446})$	$\mathbf{79.18 \pm 0.43}$ $(\mathbf{32,259})$
GCN-GraphAdaMix ($K = 4$)	80.53 ± 1.18 $(92,252)$	70.08 ± 1.59 $(237,464)$	78.42 ± 1.14 $(32,268)$

Here we examine whether increasing the number of parameters to be comparable to GraphAdaMix would achieve the same performance. Not suprisingly, this could degrade the performance due to overfitting as we can see in Table 6. However, similar observation can also be applied to GCN-GraphAdaMix. When setting $K = 3$ and $K = 4$, the performance of GCN-GraphAdaMix decline more substantially. We suspect that this is due to the fact that the mixture parameters π overfit the training data. Hence, regularizing the mixture parameters will be one of the important future work for GraphAdaMix.

D.5 The performance and the effect of ρ and K in unsupervised learning settings

Table 7: Additional average test accuracy results for unsupervised node representation learning.

	Cora	Citeseer	Pubmed
BGRL	80.6	60.4	83.5
BGRL-GraphAdaMix ($K = 2$)	**82.1**	**63.3**	85.0
BGRL-GraphAdaMix ($K = 3$)	81.7	62.0	**85.3**
BGRL-GraphAdaMix ($K = 4$)	80.1	58.8	84.2

The BGRL-GraphAdaMix model depends on the regularization parameter ρ and the number of GNNs K. Figure 6 illustrates the effect of these parameters on the performance of the BGRL-GraphAdaMix model.

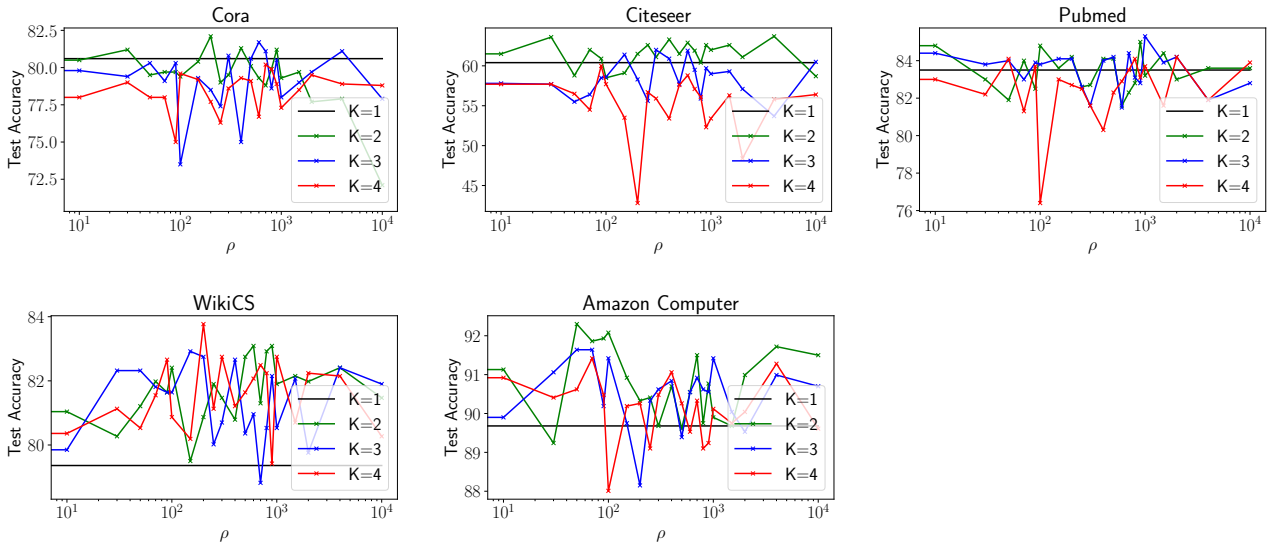

Figure 6: Influence of K and ρ on node classification accuracy using BGRL-GraphAdaMix. Markers denote mean testing classification accuracy over 10 runs. Shaded areas denote standard error. K is selected from $\{1, 2, 3, 4\}$ and ρ is selected from $\{0, 10, 30, 50, 70, 90, 100, 150, 200, 250, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000, 4000, 10000\}$. **Best viewed in color.**

Influence of ρ We train the BGRL-GraphAdaMix model on the Cora, Citeseer, Pubmed, WikiCS, and Amazon Computer datasets. We select ρ from $\{0, 10, 30, 50, 70, 90, 100, 150, 200, 250, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000, 4000, 10000\}$. In this unsupervised learning setting, we can see from Fig. 6 that ρ does not influence much on the classification accuracy. We suspect that this is due to the randomness of the dataset split.

Influence of K We select K from $\{1, 2, 3, 4\}$, where the BGRL-GraphAdaMix with $K = 1$ is equivalent to the vanilla BGRL model. Unlike GCN-GraphAdaMix, we found that K does not have much influence on the performance of the BGRL-GraphAdaMix model in most datasets. Again, we suspect that this is due to the randomness of the dataset split. The only exception here is the Citeseer dataset, where we found that setting $K = 2$ gives better overall performance than other values of K. For Citeseer, WikiCS and Amazon Computer, we found that BGRL-GraphAdaMix consistently outperforms the vanilla BGRL model ($K = 1$), which implies that there could be hidden factors reside in these datasets.