TUM School of Computation, Information and Technology
Technische Universität München

Unsupervised Learning of Network Embedding with Variational Autoencoder Framework

Rayyan Ahmad Khan

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:

Prof. Dr.-Ing. Wolfgang Kellerer

Prüfer der Dissertation:

1. Priv.-Doz. Dr. rer. nat. Martin Kleinsteuber
2. Prof. Dr.-Ing. Klaus Diepold

Die Dissertation wurde am 06.04.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 29.09.2023 angenommen.

Rayyan Ahmad Khan. Unsupervised Learning of Network Embedding with Variational Autoencoder Framework. Dissertation, Technische Universität München, Munich, Germany, 2023.

Acknowledgements

I would like to start by thanking my parents, who were always willing to sacrifice their present for my future. The lessons of diligence and perseverance, taught by them, have always been a guide throughout my life in general and my academic journey in particular.

Afterward, I owe my deepest gratitude to my supervisor, Martin. His continuous encouragement, optimism, belief and support helped me a lot in moving forward. He maintained his supervision very balanced by providing valuable time and guidance whenever needed and allowing me the freedom to explore different research areas otherwise.

Next, I would like to thank my research collaborators, especially Muhammad Umer Anwaar. We started this research journey almost in parallel. Therefore, having a similar timeline and research aptitude made it really fun for me to discuss different research ideas with him. In addition, I am also thankful to the students who helped me throughout my journey, especially Mr. Omran Kaddah.

Towards the end, I would like to express my gratitude to Unite Services GmbH & Co. KG for providing a research environment full of friendliness and freedom. I thank the members of the Analytics teams for providing valuable suggestions along the way.

München, Date TBD Rayyan Ahmad Khan

Abstract

Unsupervised Representation Learning aims to reduce the dimensionality of given data in an unsupervised fashion while retaining the important information required for downstream tasks such as classification, clustering, and visualization, etc. Such representations not only alleviate the issue of feature engineering but also make the data more usable for other Machine Learning algorithms. Variational Autoencoder (VAE) is one of the most well-known generative models, employed for unsupervised learning of the representation of the input data. In this work, we study VAE in the context of graph-structured data and examine its utility in network embedding. We limit ourselves to homophilic networks, i.e., the graphs where the nodes tend to connect to similar nodes, and vice versa.

We start by investigating Variational Graph Autoencoder (VGAE) - the extension of VAE to graph datasets - with respect to *over-pruning*, a phenomenon that limits the modeling and generative capacity of VAE in general and VGAE in particular. After reviewing the drawbacks of the current solution implemented in VGAE, we propose a model-based approach named Epitomic Variational Graph Autoencoder (**EVGAE**) to tackle the over-pruning issue such that the generative ability of VGAE is also maintained.

Graph Neural Network (GNN) layers often focus on the structural information limited to the immediate neighborhood. In this work, we look at the possibility of *explicitly* preserving the information in both the immediate and the larger neighborhood of nodes. For this, we again exploit the baseline framework of VAE for graphs and propose **Barlow Variational Graph Autoencoder** (BVGAE) - a technique that makes use of the redundancy-reduction-principle to efficiently fuse the two sources of information.

In the second half of this work, we zoom in on a sub-problem of network embedding, i.e., community-aware network embedding. We first target the homogeneous graphs and propose **J-Enc** - a variational framework for jointly encoding the network embedding and the community embeddings. The learning is constrained in such a way that connected nodes are not only "closer" to each other but also share similar community assignments. We then extend this approach to heterogeneous information networks (HINs) and propose **VaCA-HINE** for jointly learning the community embeddings and **Va**riational **C**ommunity-**A**ware **HIN E**mbedding. For both types of networks, we learn a single embedding enriched with information in immediate as well as larger neighborhoods. For homogeneous graphs, we use communities as a proxy for the larger neighborhood. For HINs, VaCA-HINE deploys contrastive modules to simultaneously utilize the information in multiple

meta-paths, thereby alleviating the meta-path selection problem - a challenge faced by many of the famous HIN embedding approaches.

Contents

List of Figures

List of Tables

1 Introduction

Unsupervised Machine Learning aims to discover patterns and relationships in the input data without the need for explicit supervisory or training labels [1]. While this field has long been an important area of Machine Learning, it has gained much more significance and popularity in recent years [2]. The main reason is the increasing availability of large, complex datasets, e.g., related to the fields of healthcare [3, 4], finance, networking [5], and natural language processing [6], etc., which are too broad and complex to be analyzed or labeled by humans.

Representation Learning [7] is one of the key research areas of Machine Learning. It deals with the distillation of important information from raw data to express it in a form that is:

- more compact, i.e., has reduced dimensionality.
- more actionable, i.e., the learned representations are general in the sense that they can be used for a multitude of downstream tasks such as clustering, classification, regression, visualization, etc.

This alleviates the problem of feature selection and also yields data in a form that is more usable for other Machine Learning algorithms. For instance, in the case of computer vision, there is no need to manually curate image features unlike before [8, 9]. Instead, the representations learned by the pre-trained computer vision models like ResNet [10] or Inception [11] are often used for specific tasks like zero-shot learning [12] and real-time object detection [13], etc. With the advent of big data, this research area has found its applications in many fields, including Natural Language Processing [14, 15], Speech Recognition and Signal Processing [16, 17], Object Recognition [18], and Recommendation Systems [14].

The models ResNet and Inception, mentioned in the example above, are trained in a supervised manner. However, Representation Learning is often employed in an unsupervised setting where it can discover useful representations of data without the need for explicit labels. **Variational Autoencoder (VAE)** [19] is one such well-known neural network model. It has a simple encoder-decoder architecture as shown in figure 1.1. The input $\boldsymbol{x}$ is fed to the encoder block which learns the parameters (μ, σ) of a Gaussian distribution $\mathcal{N}(\mu, \sigma)$. Using these parameters, we sample the latent representation $\boldsymbol{z}$ from $\mathcal{N}(\mu, \sigma)$. These samples are fed to the decoder block which attempts to reconstruct $\hat{\boldsymbol{x}}$ such that $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ are similar. In summary, VAE aims to learn the representations of the input dataset such that:

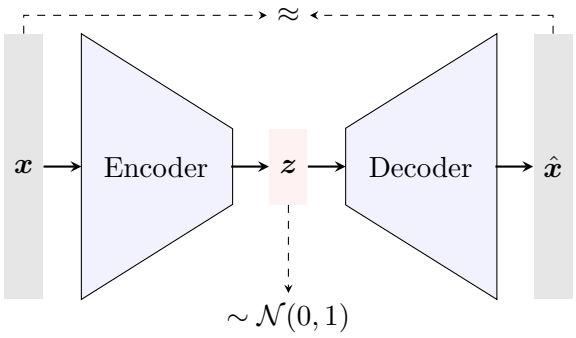


Figure 1.1: Architecture of Variational Autoencoder. The input $\boldsymbol{x}$ is encoded into latent embedding $\boldsymbol{z}$, which is aimed to be normally distributed. This embedding is then fed to the decoder, which reconstructs $\hat{\boldsymbol{x}}$ such that $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ are similar. The blue blocks are the learnable parts of VAE. The solid arrows show the flow of data, while the dashed arrows represent the parts of the objective that VAE aims to achieve.

- The latent embeddings have standard Gaussian distribution. This is achieved by minimizing Kullback Leibler (KL) divergence between $\mathcal{N}(\mu, \sigma)$ and the standard Gaussian.
- The reconstructed data is similar to the input data. This is achieved by minimizing the cross-entropy between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$. Since the dimension of $\boldsymbol{z}$ is set much lower than that of $\boldsymbol{x}$, minimizing this loss ensures that the latent embedding only distills the information required for efficient reconstruction and leaves out the rest.

There is also a simple non-variational counterpart of VAE, known as Autoencoder, where $\boldsymbol{x}$ is directly mapped to $\boldsymbol{z}$. There are, however, many advantages of mapping the input $\boldsymbol{x}$ to the parameters (μ, σ) instead of $\boldsymbol{z}$ directly. By using the distribution samples instead of a deterministic map, we add a natural regularization in the latent embeddings, thereby alleviating the problem of potential overfitting. It also enables a better exploration of the latent space, because we can sample multiple $\boldsymbol{z}$ per $\boldsymbol{x}$. Moreover, VAE gets its generative ability because of this mapping, i.e., once the model is trained, we can generate representations, similar to encoded input, by simply sampling from $\mathcal{N}(\mu, \sigma)$. Hence, by feeding these samples to the decoder, we can generate data similar to the input data on which VAE was trained. Because of the highly intuitive underlying mathematics and simple yet extensible architecture, VAE has been employed in various domains and applications.

All the methodologies, presented in this work, employ VAE as a base model for representation learning. In addition, we focus on the graph-domain datasets. **Graphs** are flexible data structures used to model complex relations in a myriad of real-world phenomena such as biological and social networks, chemistry, knowledge graphs, and many others [20–24]. Recent years have seen a remarkable interest in the field of graph analysis in general and unsupervised network embedding

learning in particular. **Network embedding** learning aims to project the graph nodes into a continuous low-dimensional vector space such that the structural properties and semantics of the network are preserved [25, 26]. The quality of the embedding is determined by downstream tasks such as transductive node classification and clustering. Recently the field of deep learning has seen a noticeable growth in the interest in graph-related problems mainly because of the increase in the computational power and the capacity of graphs to model complex relations between objects. Deep learning applications related to graphs include but are not limited to link prediction, node classification, clustering [27] and recommender systems [28–30]. In addition to the datasets derived directly from physical phenomena, there are many other areas where graphs have found their applications, e.g., in knowledge graphs for reasoning [31], in graph databases [32], and in Natural Language Processing [33, 34].

The diversity in graph datasets draws us to study them in the light of **homophily** - the observed tendency of "like to associate with like" or "Similarity breeds connection". In the context of graphs, this means that the nodes in a graph tend to be connected to other nodes that are similar to themselves. The principle of homophily was first hypothesized in [35] for social networks. However, it has shown to be an important empirical characteristic of a variety of graph-based datasets such as co-authorship networks [36], peer influence processes [37], and e-commerce networks [38]. It is worth noticing that not all graphs exhibit this characteristic. For instance, in a knowledge graph about movies, the movies of different genres are more likely to be connected if they have little in common but are watched by many people. As another example, let us consider a transportation network where nodes represent modes of communication e.g., airports, bus stations, train stations, etc. In such a network, the nodes serving different modes of transport are likely to be connected, e.g., an airport might be connected to multiple bus stations and train stations. In this work, we restrict ourselves to the graphs which exhibit homophily. The information in the links of such graphs is complementary, not competing or contrasting with the information provided by the node features. It is also worth noticing that depending on the problem at hand, the above-mentioned networks can also be viewed as homophilic. For instance, one may want to jointly cluster the users and the movies based on their connections. In such a scenario, the similarity between the movies will be governed not by their genre or plot, but by the users watching them, which adds the notion of homophily to the network.

In the first half of this work, we start by studying the Variational Graph Autoencoder (VGAE) model [39] in chapter 2. VGAE is the extension of VAE to graph datasets where the authors propose a simple yet intuitive architecture by incorporating Graph Convolutional Network (GCN) [40] layers for learning the network embedding such that the adjacency matrix can be reconstructed from the node representations. GCN layers are designed to learn from both the node attributes and the links between them. There is, however, a known issue with VAE. The KL-divergence part of the VAE objective attempts to achieve standard

Gaussian distribution for *every* dimension of the latent embedding. However, the reconstruction loss does not enforce anything on *individual* dimensions. As a result, the latent dimensions, that fail to capture enough information for reconstruction, are harshly penalized by VAE, thereby rendering them practically useless. Hence the model is not utilized to its full potential as a significant number of latent dimensions simply become *inactive*. This issue, referred to as **over-pruning**, was highlighted by Burda et al. [41] in their seminal paper [42, 43]. VGAE, being an extension of VAE, suffers from the same issue. We show that sometimes less than 20% latent dimensions remain active and the rest simply output Gaussian noise. For VAE, several solutions have been proposed to tackle this problem [44, 45]. As for the VGAE model, the authors scale down the effect of KL-divergence loss-term. While this mitigates the effect of over-pruning, it effectively turns VGAE into a non-variational model GAE because now the total loss is practically driven by the reconstruction loss only. In contrast, we take motivation from [45] to introduce a model-based approach in chapter 1 for tackling over-pruning in the graph domain. We divide the latent dimension into overlapping subsets called *epitomes* and reconstruct the adjacency matrix in a way that multiple epitomes are encouraged to participate in the reconstruction. This aids in increasing active units as epitomes compete to learn a better representation of the graph data. The resulting approach, named **Epitomic Variational Graph Autoencoder** (EVGAE) alleviates the issue of over-pruning while retaining the effect of KL-divergence loss-term.

One main reason for the easy extensibility of VAE is that its architecture is independent of the choice of encoder and decoder blocks. This is what VGAE exploits by using GCN layers in the encoder blocks. In practice we can also use other Graph Neural Network (GNN) layers, such as Graph Attention Network (GAT) [46], Graph Isomorphism Networks (GIN) [47], and GraphSAGE [48], etc. However, most neural network-based approaches focus on the structural information by limiting themselves to the immediate neighborhood. In contrast, spectral, random-walk based and matrix-factorization-based methods aim to learn the node representations by employing the proximity information which is not limited to the first-hop neighbors. Intuitively, employing the information in a larger neighborhood should improve the quality of the learned network embedding. Although we can learn from the information in the larger neighborhood by using multiple layers of GNNs, or by using the approaches like [49–51], the information is still captured in an *implicit* manner in the sense that there is no objective function to ensure the preservation of the information in the larger neighborhood. Motivated by this, in chapter 3 we study the possibility of extending the VAE architecture for learning the network embedding by simultaneously incorporating the information in the immediate as well as the larger neighborhood in an *explicit* manner. To efficiently merge the two sources of information, we take inspiration from [52], where H. Barlow hypothesized that sensory processing should aim to code highly redundant signals into statistically independent components. This

redundancy-reduction-principle has been recently applied to the image domain in Barlow Twins [53]. In this work, we use the same principle and propose **Barlow Graph Autoencoder** (BGAE) along with its variational counterpart named **Barlow Variational Graph Autoencoder** (BVGAE). Both BGAE and BVGAE aim to maximize the similarity between the embedding vectors of immediate and larger neighborhoods of a node while minimizing the redundancy between the components of these projections. As demonstrated in chapter 3, our approach performs at par with the state of the art, often outperforming its direct competitors.

In the second half of our work, we focus on a more specific sub-task of representation learning on graphs, i.e., learning representations in a community-aware manner. A community can be defined as a set of nodes that share denser connections amongst themselves compared to the other nodes of the network. Community detection, one of the major tasks in unsupervised graph analysis, aims to group the nodes into different communities. In the literature, representation learning and community detection are usually treated separately. Community detection is often treated as a downstream task to be performed using the learned network embedding. In the case of euclidean datasets, there are many studies to demonstrate that the joint learning of latent embeddings and clustering assignments yields better results compared to when these two tasks are treated independently [54–57]. We first look at this problem in chapter 4 in the context of homogeneous graphs, i.e., the graphs with a single node type. For graph datasets, some recent approaches, like CNRL [58], ComE [59], vGraph [60], etc., propose to learn the node embeddings and detect communities simultaneously in a unified framework. However, these methods learn two embeddings for each node, one used for the node representation and the other serves as the "context" of the node to aid in community detection. In contrast, in chapter 4 we argue that a single network embedding is enough for learning both the representations of the nodes as well as their contexts needed for community detection. We propose **J-Enc** - an architecture for unsupervised learning of a single community-aware networking embedding. This has advantages for downstream analysis because the analyst has to deal with only a single embedding enriched with the local information from the adjacency matrix, as well as the global information captured during simultaneous community detection. In addition, our approach is performant as well as computationally efficient.

In the last chapter, i.e., chapter 5, we extend the above-mentioned approach to Heterogeneous Information Networks (HINs), i.e., the networks where either nodes, or links, or both are of multiple types. HINs can be used to explicitly model more complex relations between multiple node types. The advantages of HINs over homogeneous networks have resulted in an increasing interest in the techniques related to HIN embedding [61]. Extension of **J-Enc** to HINs consists of two parts. Since the adjacency matrices and nodes now contain richer information, we need to plug in more generic GNN layers in the encoder block. This is achieved by employing Relational Graph Convolution Network (RGCN) [62] layers which explicitly model

heterogeneity in the nodes and edges. The second, and more important part, is to revise the sampling techniques incorporated during training. For instance, we can use random walks [63] in homogeneous graphs to get sample interactions of a node with a larger neighborhood. However, for heterogeneous networks, we need to devise this more carefully because different edge types can have different weights. As an example, if a node v is connected to N nodes of type A and a single node w of type B, a naive sampling will visit the connection $v \rightarrow w$ only $\frac{1}{1+N}$ times, which can yield poor results if the relation $v \rightarrow w$ holds high significance. To model this explicitly, we use the concept of *meta-path*. A meta-path is a tuple of node types, which serves as a template to dictate the order in which the relations between the nodes are sampled. In practice, the quality of the HIN embedding is greatly affected by the choice of the meta-path [64]. Hence, one needs to be careful in selecting which meta-path to choose for training. This needs domain knowledge, which can be subjective and expensive. Recently, some approaches have been proposed to fuse information from predefined meta-paths [65–67]. In this work, we propose **VaCA-HINE** for learning **Va**riational **C**ommunity-**A**ware **HIN E**mbedding. VaCA-HINE learns a single network embedding, just like in the case of J-Enc. Moreover, it allows the usage of multiple meta-paths, thereby alleviating the problem of meta-path selection. To embed the high-order information from multiple meta-paths, we employ *Contrastive Learning* - a ML technique that involves training a model to recognize the differences between two or more groups of data. Using this approach, we preserve high-order HIN semantics by discriminating between real and corrupted instances of different meta-paths. The network embedding, learned by VaCA-HINE, contains the information in pairwise relations, the community assignments, and the high-order HIN structure, making it a suitable candidate for downstream tasks.

1.1 List of publications

This work includes the following publications*:

1. **Khan, R.A.***, Anwaar, M.U.* and Kleinsteuber, M., 2021, January. Epitomic variational graph autoencoder. In 2020 25th International Conference on Pattern Recognition (ICPR) (pp. 7203-7210). IEEE.

2. **Khan, R.A.*** and Kleinsteuber, M., 2021. Barlow Graph Auto-Encoder for Unsupervised Network Embedding. arXiv preprint arXiv:2110.15742. Accepted at the 26th International Conference on Artificial Intelligence and Statistics (AISTATS).

3. **Khan, R.A.***, Anwaar, M.U.*, Kaddah, O.*, Han, Z*. and Kleinsteuber, M., 2021, September. Unsupervised Learning of Joint Embeddings for Node

* denotes equal contribution.

Representation and Community Detection. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD) (pp. 19-35). Springer, Cham.

4. **Khan, R.A.*** and Kleinsteuber, M., 2022, February. Cluster-Aware Heterogeneous Information Network Embedding. In Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM) (pp. 476-486).

The following were not made a part of this thesis:

1. Anwaar, M.U.*, **Khan, R.A.***, Pan, Z. and Kleinsteuber, M., 2021, October. A Contrastive Learning Approach for Compositional Zero-Shot Learning. In Proceedings of the 2021 International Conference on Multimodal Interaction (pp. 34-42).

2. Anwaar, M.U.*, Han, Z.*, Arumugaswamy, S.*, **Khan, R.A.***, Weber, T., Qiu, T., Shen, H., Liu, Y. and Kleinsteuber, M., 2022, October. On Leveraging the Metapath and Entity Aware Subgraphs for Recommendation. In Proceedings of the 1st Workshop on Multimedia Computing towards Fashion Recommendation (pp. 3-10)

3. **Khan, R.A.***, Amjad, R.A.* and Kleinsteuber, M., 2018. Extended affinity propagation: global discovery and local insights. arXiv preprint arXiv:1803.04459.

1.2 Notation

Description	Symbol
$M \times N$ matrix $\mathbf{X}$ of real numbers	$\mathbf{X} \in \mathbb{R}^{M \times N}$
$M \times N$ binary matrix $\mathbf{X}$	$\mathbf{X} \in \{0,1\}^{M \times N}$
i-th row of matrix $\mathbf{X}$	$\boldsymbol{x}_i$
i-th entry of a vector $\boldsymbol{c}$	c_i
Entry in i-th row and j-th column of $\mathbf{X}$	x_{ij}
Sum over all x_i	$\sum_i x_i$
Integration over the support of $f(\cdot)$	$\int f(z)dz$
Product over all x_i	$\prod_i x_i$
Function $f(\cdot)$, parameterized by θ	$f_\theta(\cdot)$
Dot product of $\boldsymbol{x}$ and $\boldsymbol{y}$	$< \boldsymbol{x}, \boldsymbol{y} >= \boldsymbol{x}^T \boldsymbol{y}$
Sample x from a distribution $q(x)$	$x \sim q(x)$
Probability of a sample x w.r.t. the distribution $q(\cdot)$	$q(x)$
Conditional probability of x given y w.r.t. the distribution $q(\cdot)$	$q(x\|y)$
Expectation of $f(\cdot)$ w.r.t. all involved random variables.	$\mathbb{E}\{f(\cdot)\}$
Expectation of $f(x)$ w.r.t. distribution $q(\cdot)$.	$\mathbb{E}_q\{f(x)\} = \mathbb{E}_{x \sim q(x)}\{f(x)\}$
Kullback-Leibler divergence between the distributions $p(\cdot)$ and $q(\cdot)$	$D_{\mathrm{KL}}(p\|\|q)$

Table 1.1: Notation and symbols used throughout the this work.

Part I

General Unsupervised Network Embedding

2 Epitomic Variational Graph Autoencoder

Based on the following peer-reviewed publication:

Khan, R.A.*, *Anwaar, M.U.** and Kleinsteuber, M., 2021, January. Epitomic variational graph autoencoder. In 2020 25th International Conference on Pattern Recognition (ICPR) (pp. 7203-7210). IEEE.*

Kipf and Welling [39] introduced variational graph autoencoder (VGAE) by extending the variational autoencoder (VAE) model [19]. Like VAE, VGAE tends to achieve the following two competitive objectives:

1. An approximation of input data should be possible.
2. The latent representation of input data should follow the standard Gaussian distribution.

There is, however, a well-known issue with VAE in general: The latent units, which fail to capture enough information about the input data, are harshly suppressed during training. As a result, the corresponding latent variables collapse to the prior distribution and are simply generating standard Gaussian noise. Consequently, in practice, the number of latent units, referred to as *active units*, actually contributing to the reconstruction of the input data are quite low compared to the total available latent units. This phenomenon is referred to as *over-pruning* [41–43]. Several solutions have been proposed to tackle this problem for VAEs. For instance, adding dropout can be a simple solution to achieve more active units. However, this solution adds redundancy rather than encoding more useful information with latent variables [45]. [44] proposes division of the hidden units into subsets and forcing each subset to contribute to the KL divergence. [42] uses KL cost annealing to activate more hidden units. [45] uses a model-based approach where latent units are divided into subsets with only one subset penalized for a certain data point. These subsets also share some latent variables which help in reducing the redundancy between different subsets.

VGAE, being an extension of VAE for graph datasets, is also susceptible to the over-pruning problem. This greatly reduces the modeling power of pure VGAE and undermines its ability to learn diverse and meaningful latent representations As demonstrated in detail in section 2.2. To suppress this issue, the authors of [39] simply reduce the weight of the second objective by a number of nodes in training

data. For instance, PubMed dataset[1] has ∼20k nodes, so the second objective is given 20,000 times less weight than the first objective. Since the second objective is the one enforcing standard gaussian distribution for the latent variables, reducing its weight adversely affects the generative ability of VGAE and effectively reduces it to non-variational graph autoencoder. We discuss this further in section 2.3.

In this work, we refer to VGAE without any weighted objectives as *pure VGAE* to distinguish it from VGAE [39]. In order to attain good generative ability and mitigate over-pruning, we adopt a model-based approach called epitomic VGAE (EVGAE). Our approach is motivated by a solution proposed for tackling the over-pruning problem in VAE [45]. We consider our model to consist of multiple sparse VGAE models, called *epitomes*, that share the latent space such that for every graph node only one epitome is forced to follow the prior distribution. This results in a higher number of active units as epitomes compete to learn a better representation of the graph data.

Our main contributions are summarized below:

- We identify that VGAE [39] has poor generative ability due to the extra factor in the training objective.
- We show that pure VGAE (without any weighted objectives) suffers from the over-pruning problem.
- We propose a true variational model EVGAE that not only achieves better generative ability than VGAE but also mitigates the over-pruning issue.

2.1 Pure Variational Graph Autoencoder

Given an undirected and unweighted graph $\mathcal{G}$ consisting of N nodes $\{\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_N}\}$ with each node having F features. We assume that the information in nodes and edges can be jointly encoded in a D dimensional latent space such that the respective latent variables $\{\boldsymbol{z_1}, \boldsymbol{z_2}, \cdots, \boldsymbol{z_N}\}$ follow the standard Gaussian distribution. These latent variables are stacked into a matrix $\boldsymbol{Z} \in \mathbb{R}^{N \times D}$. For reconstructing the input data, this matrix is then fed to the decoder network $p_\theta(\mathcal{G}|\boldsymbol{Z})$ parameterized by θ. The assumption on latent representation allows the trained model to generate new data, similar to the training data, by sampling from the prior distribution. Following VAE, the joint distribution can be written as

$$p(\mathcal{G}, \boldsymbol{Z}) = p(\boldsymbol{Z}) p_\theta(\mathcal{G}|\boldsymbol{Z}), \tag{2.1}$$

[1]PubMed is a citation dataset [68], widely used in deep learning for graph analysis. Details of the dataset are given in experiments section 2.5.1

where

$$p(\boldsymbol{Z}) = \prod_{i=0}^{N} p(\boldsymbol{z}_i) \tag{2.2}$$

$$p(\boldsymbol{z}_i) = \mathcal{N}(\mathbf{0}, \mathrm{diag}(\mathbf{1}))\ \forall i. \tag{2.3}$$

For an unweighted and undirected graph $\mathcal{G}$, we follow [39] and restrict the decoder to reconstruct only edge information from the latent space. The edge information can be represented by an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ where a_{ij} refers to the element in i-th row and j-th column. If an edge exists between node i and j, we have $\mathrm{a}_{ij} = 1$. Thus, the decoder is given by

$$p_\theta(\boldsymbol{A}|\boldsymbol{Z}) = \prod_{(i,j)=(1,1)}^{(N,N)} p_\theta(\mathrm{a}_{ij} = 1|\boldsymbol{z_i}, \boldsymbol{z_j}), \tag{2.4}$$

with

$$p_\theta(\mathrm{a}_{ij} = 1|\boldsymbol{z_i}, \boldsymbol{z_j}) = \mathrm{sigmoid}(< \boldsymbol{z_i}, \boldsymbol{z_j} >), \tag{2.5}$$

where $< .\ ,\ . >$ denotes dot product. The training objective should be such that the model is able to generate new data and recover graph information from the embeddings simultaneously. For this, we aim to learn the free parameters of our model such that the log probability of $\mathcal{G}$ is maximized i.e.

$$\begin{aligned} log\Big(p(\mathcal{G})\Big) &= log\Big(\int p(\boldsymbol{Z}) p_\theta(\mathcal{G}|\boldsymbol{Z})\ d\boldsymbol{Z}\Big) \\ &= log\Big(\int \frac{q_\phi(\boldsymbol{Z}|\mathcal{G})}{q_\phi(\boldsymbol{Z}|\mathcal{G})} p(\boldsymbol{Z}) p_\theta(\mathcal{G}|\boldsymbol{Z})\ d\boldsymbol{Z}\Big) \\ &= log\Big(\mathbb{E}_{\boldsymbol{Z} \sim q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big\{\frac{p(\boldsymbol{Z}) p_\theta(\mathcal{G}|\boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big\}\Big), \end{aligned} \tag{2.6}$$

where $q_\phi(\boldsymbol{Z}|\mathcal{G})$, parameterized by ϕ, models the recognition network for approximate posterior inference. It is given by

$$q_\phi(\boldsymbol{Z}|\mathcal{G}) = \prod_{i}^{N} q_\phi(\boldsymbol{z_i}|\mathcal{G}) \tag{2.7}$$

$$q_\phi(\boldsymbol{z_i}|\mathcal{G}) = \mathcal{N}\Big(\boldsymbol{\mu_i}(\mathcal{G}), \mathrm{diag}(\boldsymbol{\sigma_i^2}(\mathcal{G}))\Big) \tag{2.8}$$

where $\boldsymbol{\mu}_i(.)$ and $\boldsymbol{\sigma}_i^2(.)$ are learned using graph convolution networks*(GCN)* [40] and samples of $q_\phi(\boldsymbol{Z}|\mathcal{G})$ are obtained from mean and variance using reparameterization trick [19].

In order to ensure computational tractability, we use Jensen's Inequality [69] to get the ELBO bound of equation 2.6. i.e.

$$log\Big(p(\mathcal{G})\Big) \geq \mathbb{E}_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{Z})p_\theta(\mathcal{G}|\boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big)\Big\} \tag{2.9}$$

$$= \mathbb{E}_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big\{log\Big(p_\theta(\mathcal{G}|\boldsymbol{Z})\Big)\Big\}$$

$$+ \mathbb{E}_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big)\Big\} \tag{2.10}$$

$$= -\text{BCE} - D_{KL}\Big(q_\phi(\boldsymbol{Z}|\mathcal{G})||p(\boldsymbol{Z})\Big) \tag{2.11}$$

where BCE denotes binary cross-entropy loss between input edges and the reconstructed edges. The Kullback-Leibler (KL) divergence is denoted by D_{KL}. By using equation 2.2, equation 2.3, equation 2.7 and equation 2.8, the loss function of pure VGAE can be formulated as negative of equation 2.11 i.e.

$$L = \text{BCE} + \sum_{i=1}^{N} D_{KL}\Big(\mathcal{N}\Big(\boldsymbol{\mu}_i(\mathcal{G}), \boldsymbol{\sigma}_i^2(\mathcal{G})\Big) \;||\; \mathcal{N}(\mathbf{0}, \text{diag}(\mathbf{1}))\Big) \tag{2.12}$$

2.2 Over-pruning in pure VGAE

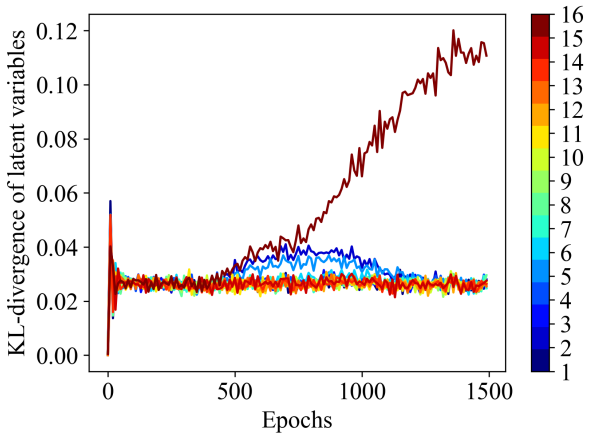
Burda et al. [41] showed that the learning capacity of VAE is limited by *over-pruning*. Several other studies [42–45] confirm this and propose different remedies for the over-pruning problem. They hold the KL-divergence term in the loss function of VAE responsible for over-pruning. This term forces the latent variables to follow the standard Gaussian distribution. Consequently, those variables which fail to encode *enough* information about input data are harshly penalized. In other words, if a latent variable is contributing little to the reconstruction, the variational loss is minimized easily by "turning off" the corresponding hidden unit. Subsequently, such variables simply collapse to the prior, i.e. generate standard Gaussian noise. We refer to the hidden units contributing to the reconstruction as *active units* and the turned-off units as *inactive units*. The activity of a hidden unit u was quantified by Burda et al. [41] via the statistic

$$A_u = Cov_x(\mathbb{E}_{u\sim q(u|x)}\{u\}). \tag{2.13}$$

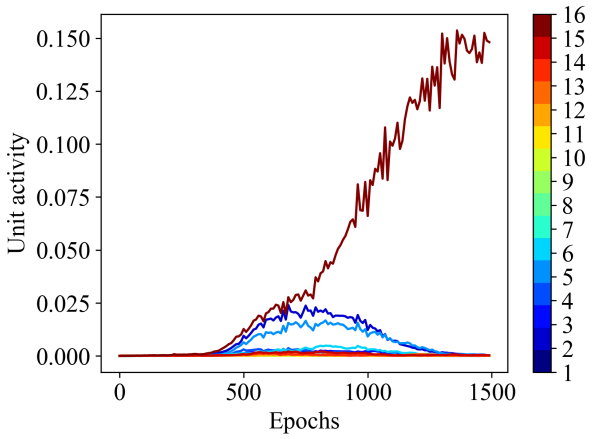
A hidden unit u is said to be *active* if $A_u \geq 10^{-2}$.

VGAE is an extension of VAE for graph data and the loss function of both models contains the KL-divergence term. Consequently, pure VGAE inherits the over-pruning issue. We verify this by training VGAE with equation 2.12 on

Cora dataset[2]. We employ the same graph architecture as Kipf and Welling [39]. The mean and log-variance of 16-dimensional latent space are learned via Graph Convolutional Networks [40]. From figure 2.1(a), we observe that 15 out of 16 latent variables have KL-divergence around 0.03, indicating that they are very closely matched with standard Gaussian distribution. Only one latent variable has managed to diverge to encode the information required by the decoder for the reconstruction of the input.



(a) KL-divergence of latent variables in pure VGAE



(b) Unit activity of 16 hidden units in pure VGAE

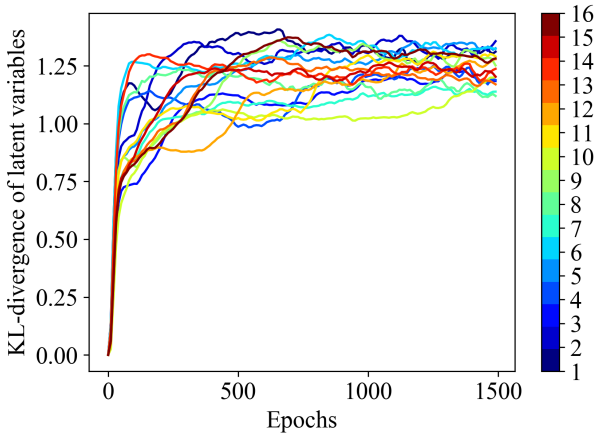
Figure 2.1: (a) show that only one out of 16 hidden units is actively encoding input information required for the reconstruction. This is confirmed by the plot of unit activity in (b).

In other words, the pure VGAE model is using only one variable for encoding the input information while the rest 15 latent variables are not learning anything about the input. These 15 latent variables collapse to the prior distribution and are simply generating standard Gaussian noise. Figure 2.1(b) shows the activity of

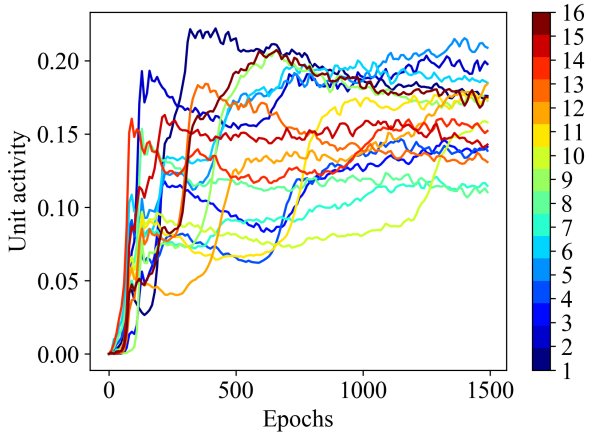
[2]Details of Cora dataset are given in experiments section 2.5.1

hidden units as defined in equation 2.13. It is clear that only one unit is *active*, which corresponds to the latent variable with the highest KL-divergence in the figure 2.1(a). All others units have become *inactive* and are not contributing to learning the reconstruction of the input. This verifies the existence of *over-pruning* in the pure VGAE model.

2.3 VGAE: Sacrificing Generative Ability for Handling Over-pruning



(a) KL-divergence of latent variables: VGAE ($\beta \approx 0.0003$ [39])



(b) Unit activity of 16 hidden units: VGAE ($\beta \approx 0.0003$ [39])

Figure 2.2: All the hidden units are active but KL-divergence is quite high, indicating poor matching of learned distribution with prior, consequently affecting the generative ability of the model.

Kipf and Welling's VGAE [39] employed a simple way to get around the over-pruning problem by adding a penalty factor to the KL-divergence in equation 2.12.

That is

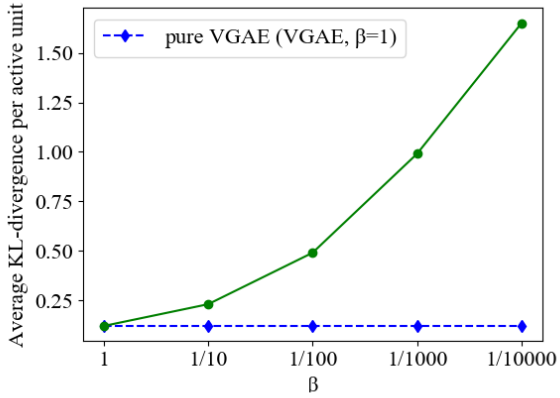
$$L = BCE + \beta \; D_{KL}\Big(q(\boldsymbol{Z}|\mathcal{G})||p(\boldsymbol{Z})\Big). \tag{2.14}$$

But a consequence of using the penalty factor β is the poor generative ability of VGAE. We verify this by training VGAE on the Cora dataset with varying β in equation 2.14. We call the penalty factor β, as the loss of βVAE [70, 71] has the same factor multiplied with its KL-divergence term. Specifically, in βVAE, $\beta > 1$ is chosen to enforce better distribution matching. Conversely, smaller β is selected for relaxing the distribution matching, i.e. the latent distribution is allowed to be more different than the prior distribution. This enables latent variables to learn better reconstruction at the expense of the generative ability. In the degenerate case, when $\beta = 0$, the VGAE model is reduced to non-variational graph autoencoder (GAE). VGAE as proposed by Kipf and Welling [39] has the loss function similar to βVAE with β chosen as the reciprocal of the number of nodes in the graph. As a result β is quite small i.e. $\sim$ 0.0001-0.00001.

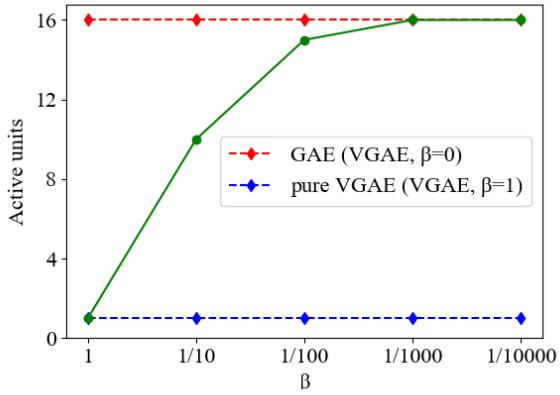
Figure 2.2 shows the KL-divergence and hidden unit activity for original VGAE [39] model. We observe that all the hidden units are active, i.e. $A_u \geq 10^{-2}$. However, the value of KL-divergence is quite high for all latent variables, indicating poor matching of $q_\phi(\boldsymbol{Z}|\mathcal{G})$ with the prior distribution. This adversely affects the generative ability of the model. Concretely, the variational model is supposed to learn such a latent representation that follow standard Gaussian (prior) distribution. Such high values of KL-divergence implies that the learned distribution is not standard gaussian. The reason is that the KL-divergence term in equation 2.14 was responsible for ensuring that the posterior distribution being learned follows standard Gaussian distribution. VGAE [39] model assigns too small weight ($\beta =$ 0.0003) to the KL-divergence term. Consequently, when new samples are generated from standard gaussian distribution $p(\boldsymbol{Z})$ and then passed through the decoder $p_\theta(\boldsymbol{A}|\boldsymbol{Z})$, we get quite different output than the graph data used for training.

Figure 2.3 shows that Kipf and Welling's [39] approach to dealing with over-pruning makes VAGE similar to its non-variational counterpart i.e. graph autoencoder (GAE). As β is decreased, VGAE model learns to give up on the generative ability and behaves similarly to GAE. This can be seen in figure 2.3 (a), where the average KL-divergence per active hidden unit increases drastically as β becomes smaller. On the other hand, we observe from figure 2.3 (b) that decreasing β results in a higher number of active hidden units till it achieves the same number as GAE.

We conclude that as the contribution of KL-divergence is penalized in the loss function (equation 2.14), the VGAE model learns to sacrifice the generative ability for avoiding over-pruning. Conversely, VGAE handles the over-pruning problem by behaving like a non-variational model GAE [72].



(a) Change in the active units of original VGAE [39]



(b) Change in the Average KL-divergence per active unit

Figure 2.3: Effect of varying β on original VGAE

2.4 Epitomic Variational Graph Autoencoder

We propose epitomic variational graph autoencoder (EVGAE) which generalizes and improves the VGAE model. EVGAE not only successfully mitigates the over-pruning issue of pure VGAE but also attains better generative ability than VGAE [39]. The motivation comes from the observation that for a certain graph node, a subset of the latent space suffices to yield good reconstruction of edges. Yeung et al. [45] proposed a similar solution for tackling the over-pruning problem in VAE. We assume M subsets of the latent space called *epitomes*. They are denoted by $\{\mathcal{D}_1, \cdots, \mathcal{D}_M\}$. Furthermore, it is ensured that every subset shares some latent variables with at least one other subset. We penalize only one epitome

for an input node. This encourages other epitomes to be active. Let y_i denote a discrete random variable that decides which epitome is active for a node i. For a given node, the prior distribution of y_i is assumed to be uniform over all the epitomes. $\boldsymbol{y}$ represents the stacked random vector for all N nodes of the graph. So:

$$p(\boldsymbol{y}) = \prod_i p(y_i) \tag{2.15}$$

$$p(y_i) = \mathcal{U}(1, M), \tag{2.16}$$

where $\mathcal{U}(\cdot)$ denotes uniform distribution.

Let $\boldsymbol{E} \in \mathbb{R}^{M \times D}$ denote a binary matrix, where each row represent an epitome and each column represents a latent variable. Figure 2.4 shows $\boldsymbol{E}$ with $M = 8$ and $D = 16$ in a D-dimensional latent space. The grayed squares of r^{th} row show the latent variables which constitute the epitome $\mathcal{D}_r$. We denote r^{th} row of $\boldsymbol{E}$ by $\boldsymbol{e}_r$.

The variational model of EVGAE is the same as VGAE i.e.:

$$p(\mathcal{G}, \boldsymbol{Z}) = p(\boldsymbol{Z}) p_\theta(\mathcal{G}|\boldsymbol{Z}) \tag{2.17}$$

The corresponding approximate posterior is given by:

$$q_\phi(\boldsymbol{Z}, \boldsymbol{y}|\mathcal{G}) = q_\phi(\boldsymbol{y}|\mathcal{G}) q_\phi(\boldsymbol{Z}|\mathcal{G}), \tag{2.18}$$

with

$$q_\phi(\boldsymbol{y}|\mathcal{G}) = \prod_{i=1}^{N} q_\phi(y_i|\mathcal{G}) \tag{2.19}$$

$$q_\phi(y_i|\mathcal{G}) = \mathrm{Cat}(\boldsymbol{\pi}_i(\mathcal{G})) \tag{2.20}$$

$$q_\phi(\boldsymbol{Z}|\mathcal{G}) = \prod_{i}^{N} q_\phi(\boldsymbol{z}_i|\mathcal{G}) \tag{2.21}$$

$$q_\phi(\boldsymbol{z}_i|\mathcal{G}) = \mathcal{N}\Big(\boldsymbol{\mu}_i(\mathcal{G}), \mathrm{diag}(\boldsymbol{\sigma}_i^2(\mathcal{G}))\Big), \tag{2.22}$$

where $\mathrm{Cat}(\cdot)$ refers to the categorical distribution. $\boldsymbol{\pi}_i(\cdot)$, $\boldsymbol{\mu}_i(\cdot)$ and $\boldsymbol{\sigma}_i^2(\cdot)$ are learned using two-layer GCN networks. Under the assumption that $\boldsymbol{y}$ and $\mathcal{G}$ are independent, given $\boldsymbol{Z}$; the objective function is given by

$$log\Big(p(\mathcal{G})\Big) = log\Big(\int \sum_{\boldsymbol{y}} p(\boldsymbol{y}) p(\boldsymbol{Z}|\boldsymbol{y}) p_\theta(\mathcal{G}|\boldsymbol{Z})\ d\boldsymbol{Z}\Big) \tag{2.23}$$

$$= log\Big(\mathbb{E}_{(\boldsymbol{Z},\boldsymbol{y})\sim q_\phi(\boldsymbol{Z},\boldsymbol{y}|\mathcal{G})}\Big\{\frac{p(\boldsymbol{y})p(\boldsymbol{Z}|\boldsymbol{y})p_\theta(\mathcal{G}|\boldsymbol{Z})}{q_\phi(\boldsymbol{Z},\boldsymbol{y}|\mathcal{G})}\Big\}\Big) \tag{2.24}$$

$$= log\Big(\mathbb{E}_{(\boldsymbol{Z},\boldsymbol{y})\sim q_\phi(\boldsymbol{Z},\boldsymbol{y}|\mathcal{G})}\Big\{\frac{p(\boldsymbol{y})p(\boldsymbol{Z}|\boldsymbol{y})p_\theta(\mathcal{G}|\boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{G})q_\phi(\boldsymbol{y}|\mathcal{G})}\Big\}\Big). \tag{2.25}$$

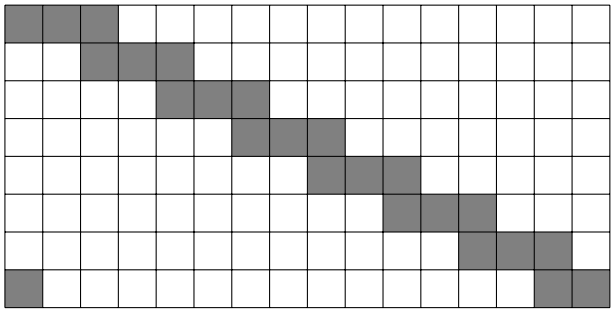


Figure 2.4: Example of eight epitomes in a 16-dimensional latent space.

By using Jensen's inequality [69], the ELBO bound for log probability becomes

$$log\Big(p(\mathcal{G})\Big) \geq \mathbb{E}_{(\boldsymbol{Z},\boldsymbol{y})\sim q_\phi(\boldsymbol{Z},\boldsymbol{y}|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{y})p(\boldsymbol{Z}|\boldsymbol{y})p_\theta(\mathcal{G}|\boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{G})q_\phi(\boldsymbol{y}|\mathcal{G})}\Big)\Big\} \tag{2.26}$$

$$\begin{aligned} &= \mathbb{E}_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big\{log\Big(p_\theta(\mathcal{G}|\boldsymbol{Z})\Big)\Big\} \\ &+ \mathbb{E}_{\boldsymbol{y}\sim q_\phi(\boldsymbol{y}|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{y})}{q_\phi(\boldsymbol{y}|\mathcal{G})}\Big)\Big\} \\ &+ \mathbb{E}_{(\boldsymbol{Z},\boldsymbol{y})\sim q_\phi(\boldsymbol{Z},\boldsymbol{y}|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{Z}|\boldsymbol{y})}{q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big)\Big\}. \end{aligned} \tag{2.27}$$

Following VGAE [39], we restrict the decoder to recover only edge information from the latent space. Hence, the decoder is the same as in equation 2.4. Thus, the first term in equation 2.27 simplifies in a similar way as in VGAE i.e. binary cross-entropy between input and reconstructed edges.

The second term in equation 2.27 is computed as:

$$\begin{aligned} \mathbb{E}_{\boldsymbol{y}\sim q_\phi(\boldsymbol{y}|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{y})}{q_\phi(\boldsymbol{y}|\mathcal{G})}\Big)\Big\} &= \mathbb{E}_{\boldsymbol{y}\sim q_\phi(\boldsymbol{y}|\mathcal{G})}\Big\{\sum_{i=1}^{N} log\Big(\frac{p(y_i)}{q_\phi(y_i|\mathcal{G})}\Big)\Big\} \\ &= \sum_{i=1}^{N} \mathbb{E}_{y_i\sim q_\phi(y_i|\mathcal{G})}\Big\{log\Big(\frac{p(y_i)}{q_\phi(y_i|\mathcal{G})}\Big)\Big\} \\ &= -\sum_{i=1}^{N} D_{KL}\Big(q_\phi(y_i|\mathcal{G})||p(y_i)\Big) \\ &= -\sum_{i=1}^{N} D_{KL}\Big(\mathrm{Cat}(\boldsymbol{\pi}_i(\mathcal{G}))||\ \mathcal{U}(1,M)\Big). \end{aligned} \tag{2.28}$$

The third term in equation 2.27 is computed as follows:

$$\begin{aligned}
&\mathbb{E}_{(\boldsymbol{Z},\boldsymbol{y})\sim q_\phi(\boldsymbol{Z},\boldsymbol{y}|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{Z}|\boldsymbol{y})}{q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big)\Big\} \\
=&\mathbb{E}_{\boldsymbol{y}\sim q_\phi(\boldsymbol{y}|\mathcal{G})}\Big\{\mathbb{E}_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{Z}|\boldsymbol{y})}{q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big)\Big\}\Big\} \\
=&\sum_{\boldsymbol{y}} q_\phi(\boldsymbol{y}|\mathcal{G})\mathbb{E}_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{Z}|\boldsymbol{y})}{q_\phi(\boldsymbol{Z}|\mathcal{G})}\Big)\Big\} \\
=&\sum_{i=1}^{N}\sum_{\boldsymbol{y}} q_\phi(\boldsymbol{y}|\mathcal{G})\mathbb{E}_{\boldsymbol{z}_i\sim q_\phi(\boldsymbol{z}_i|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{z}_i|y_i)}{q_\phi(\boldsymbol{z}_i|\mathcal{G})}\Big)\Big\} \\
=&\sum_{i=1}^{N}\sum_{y_i} q_\phi(y_i|\mathcal{G})\mathbb{E}_{\boldsymbol{z}_i\sim q_\phi(\boldsymbol{z}_i|\mathcal{G})}\Big\{log\Big(\frac{p(\boldsymbol{z}_i|y_i)}{q_\phi(\boldsymbol{z}_i|\mathcal{G})}\Big)\Big\} \\
=&-\sum_{i=1}^{N}\sum_{y_i} q_\phi(y_i|\mathcal{G})D_{KL}\Big(q_\phi(\boldsymbol{z}_i|\mathcal{G})||p(\boldsymbol{z}_i|y_i)\Big) \qquad (2.29)
\end{aligned}$$

We take motivation from [45] to compute equation 2.29 as:

$$\begin{aligned}
&-\sum_{i=1}^{N}\sum_{y_i} q_\phi(y_i|\mathcal{G})D_{KL}\Big(q_\phi(\boldsymbol{z}_i|\mathcal{G})||p(\boldsymbol{z}_i|y_i)\Big) \\
=&-\sum_{i=1}^{N}\sum_{y_i} q_\phi(y_i|\mathcal{G})\sum_{j=1}^{D}\boldsymbol{E}[y_i,j]D_{KL}\Big(q_\phi(z_i^j|\mathcal{G})||p(z_i^j)\Big) \qquad (2.30) \\
=&-\sum_{i=1}^{N}\sum_{y_i} \boldsymbol{\pi}_i(\mathcal{G})\sum_{j=1}^{D}\boldsymbol{E}[y_i,j]D_{KL}\Big(\mathcal{N}\Big(\mu_i^j(\mathcal{G}),(\sigma_i^2)^j(\mathcal{G})\Big)||\mathcal{N}(0,1)\Big), \qquad (2.31)
\end{aligned}$$

where $\boldsymbol{E}[y_i,j]$ refers to j^{th} component of epitome y_i and z_i^j denotes j^{th} component of vector $\boldsymbol{z}_i$. In equation 2.31, for each node, we sum over all the epitomes. For a given epitome, we only consider the effect of those latent variables which are selected by $\boldsymbol{E}$ for that epitome. This also implies that the remaining latent variables have the freedom to better learn the reconstruction. Consequently, EVGAE encourages more hidden units to be active without penalizing the hidden units which are contributing little to the reconstruction. The final loss function is given by:

$$\begin{aligned}
L=&\ \mathrm{BCE}+\sum_{i=1}^{N} D_{KL}\Big(\mathrm{Cat}(\boldsymbol{\pi}_i(\mathcal{G}))||\ \mathcal{U}(1,M)\Big) \\
&+\sum_{i=1}^{N}\sum_{y_i} \boldsymbol{\pi}_i(\mathcal{G})\sum_{j=1}^{D}\boldsymbol{E}[y_i,j]D_{KL}\Big(\mathcal{N}\Big(\mu_i^j(\mathcal{G}),(\sigma_i^2)^j(\mathcal{G})\Big)||\mathcal{N}(0,1)\Big). \qquad (2.32)
\end{aligned}$$

Algorithm 1: EVGAE Algorithm

Input:

- $\mathcal{G}$
- Epochs
- The matrix $\boldsymbol{E}$ to select latent variables for each epitome.

Initialize model weights; $i = 1$
while $e \leq Epochs$ **do**
  compute $\boldsymbol{\pi_i(.)}$, $\boldsymbol{\mu_i(.)}$ and $\boldsymbol{\sigma_i^2(.)}$ $\forall i$;
  compute $\boldsymbol{z}_i$ $\forall i$ by reparameterization trick;
  compute loss using equation 2.32;
  update model weights using backpropagation
end

VGAE model can be recovered from EVGAE model, if we have only one epitome consisting of all latent variables. Hence the model generalizes VGAE. The algorithm for training EVGAE is given in Algo. 1.

2.5 Experiments

2.5.1 Datasets

We compare the performance of EVGAE with several baseline methods on the link prediction task. We conduct the experiments on three benchmark citation datasets [68].

Cora dataset has 2,708 nodes with 5,297 undirected and unweighted links. The nodes are defined by 1433 dimensional binary feature vectors, divided in 7 classes.

Citeseer dataset has 3,312 nodes defined by 3703 dimensional feature vectors. The nodes are divided in 6 distinct classes. There are 4,732 links between the nodes.

PubMed consists of 19,717 nodes defined by 500 dimensional feature vectors linked by 44,338 unweighted and undirected edges. These nodes are divided in 3 classes.

2.5.2 Implementation Details and Performance Comparison

In order to ensure fair comparison, we follow the experimental setup of Kipf and Welling [39]. That is, we train the EVGAE and pure VGAE model on an incomplete version of citation datasets. Concretely, the edges of the dataset are divided in

training set, validation set and test set. Following [39], we use 85% edges for training, 5% for validation and 10% for testing the performance of the model.

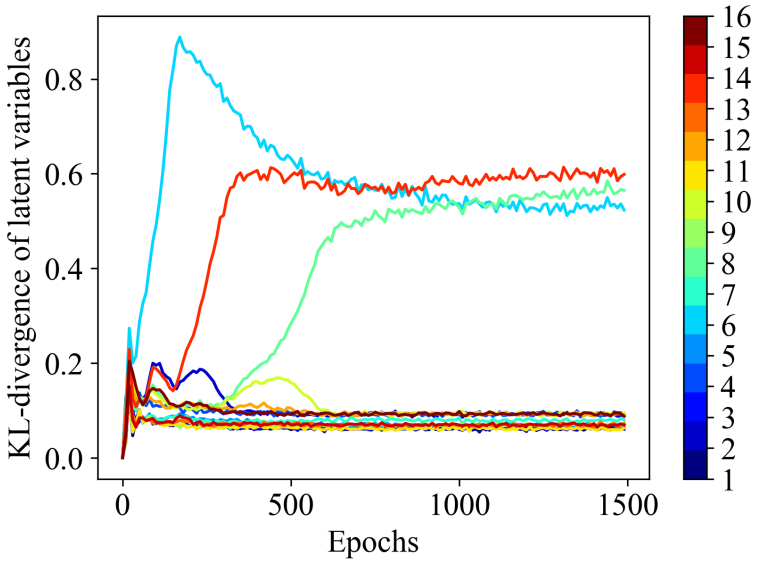
We compare the performance of EVGAE with three strong baselines, namely: VGAE [39], spectral clustering [73] and DeepWalk [63]. We also report the performance of pure VGAE (β=1) and GAE (VGAE with β=0). Since DeepWalk and spectral clustering do not employ node features; so VGAE, GAE and EVGAE have an undue advantage over them. The implementation of spectral clustering is taken from [74] with 128 dimensional embedding and for DeepWalk, the standard implementation is used [75]. For VGAE and GAE, we use the implementation provided by Kipf and Welling [39]. EVGAE also follows a similar structure with latent embedding being 512 dimensional and the hidden layer consisting of 1024 hidden units, half of which learn $\boldsymbol{\mu}_i(.)$ and the other half for learning log-variance. We select 256 epitomes for all three datasets. Each epitome enforces three units to be active, while sharing one unit with neighboring epitomes. This can also be viewed as an extension of the matrix shown in figure 2.4. Adam [76] is used as optimizer with learning rate $1e^{-3}$. Further implementation details of EVGAE can be found in the code [77].

For evaluation, we follow the same protocols as other recent works [39, 63, 73]. That is, we measure the performance of models in terms of area under the ROC curve (AUC) and average precision (AP) scores on the test set. We repeat each experiment 10 times in order to estimate the mean and the standard deviation in the performance of the models.

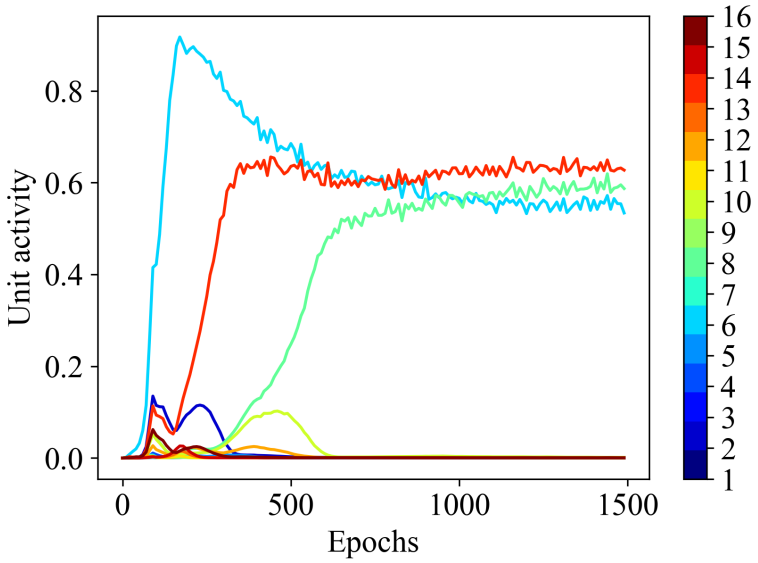
Method	**Cora**		**Citeseer**		**PubMed**	
	AUC	AP	AUC	AP	AUC	AP
DeepWalk	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01	84.4 ± 0.00	84.1 ± 0.0
Spectral Clustering	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01	84.2 ± 0.02	87.8 ± 0.01
GAE (VGAE [39] with $\beta = 0$)	91.0 ± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	96.4 ± 0.00	96.5 ± 0.0
VGAE [39] ($\beta \sim 10^{-4} - 10^{-5}$)	91.4 ± 0.01	92.6 ± 0.01	90.8 ± 0.02	92.0 ± 0.02	94.4 ± 0.02	94.7 ± 0.0
pure VGAE ($\beta = 1$)	79.44 ± 0.03	80.51 ± 0.02	77.08 ± 0.03	79.07 ± 0.02	82.79 ± 0.01	83.88 ± 0.01
EVGAE ($\beta = 1$)	$\mathbf{92.96 \pm 0.02}$	$\mathbf{93.58 \pm 0.03}$	$\mathbf{91.55 \pm 0.03}$	$\mathbf{93.24 \pm 0.02}$	$\mathbf{96.80 \pm 0.01}$	$\mathbf{96.91 \pm 0.02}$

Table 2.1: Results of link prediction on citation datasets

We can observe from Table 2.1 that the results of EVGAE are competitive or slightly better than other methods. We also note that the performance of variational method pure VGAE is quite bad as compared to our variational method EVGAE. Moreover, the performance of methods with no or poor generative ability (GAE and VGAE [39] with $\beta \sim 10^{-4} - 10^{-5}$) is quite similar.



(a) KL-divergence of latent variables in EVGAE



(b) Unit activity of 16 hidden units of EVGAE

Figure 2.5: Three hidden units are active and KL-divergence of corresponding latent variables is quite low compared to figure 2.2(a), indicating a good matching of learned distribution with prior, consequently improving the generative ability of the model.

2.5.3 EVGAE: Over-pruning and Generative Ability

We now show the learning behavior of EVGAE model on our running example of Cora dataset. We select 8 epitomes, each dictating three hidden units to be active. The configuration is shown in figure 2.4. Figure 2.5 shows the evolution of KL-divergence and unit activity during training of EVGAE model. By comparing this figure with pure VGAE (figure 2.1), we can observe that EVGAE has more active hidden units. This demonstrates that our model is better than pure VGAE at mitigating the over-pruning issue.

On the other hand, if we compare it to VGAE [39](figure 2.2), we observe EVGAE have less active units in comparison. But KL-divergence of the latent variables for VGAE is greater than 1 for all the latent variables (figure 2.2(a)). This implies that the latent distribution is quite different from the prior distribution (standard gaussian). In contrast, we observe from figure 2.5(a) that EVGAE has KL-divergence around 0.1 for 13 latent variables and approximately 0.6 for remaining 3 latent variables. This reinforces our claim that VGAE achieves more active hidden units by excessively penalizing the KL-term responsible for generative ability.

In short, although EVGAE has less active units, the distribution matching is better compared to VGAE. VGAE is akin to GAE due to such low weightage to KL-term, i.e. $\beta = 0.0003$.

2.5.4 Impact of Latent Space Dimension

We now look at the impact of latent space dimension on the number of active units and average KL-divergence per active unit. We plot the active units for dimensions $D \in \{16, 32, 64, 128, 256, 512\}$. Figure 2.6 presents an overview of this impact on our running example (Cora dataset). For all values of D, the number of epitomes is set to $\frac{D}{2}$ and one unit is allowed to overlap with neighboring epitomes. Similar to the configuration in figure 2.4 for $D = 16$. It is to be noted that we kept same configuration of epitomes for consistency reasons. Choosing a different configuration of epitomes does not affect the learning behavior of EVGAE.

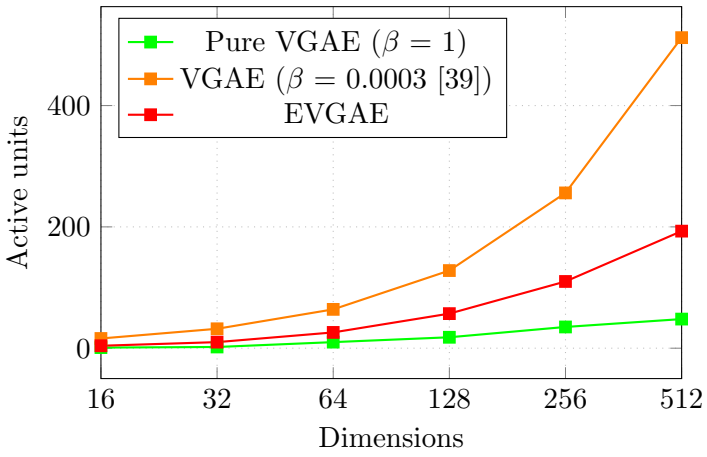
It can be observed that the number of active units is quite less compared to the available units for VGAE with $\beta = 1$ (pure VGAE). Concretely, for $D = 512$ only 48 units are active. This shows that the over-pruning problem persists even in high dimensional latent space.

Now we observe the behavior of VGAE with $\beta = N^{-1}$ as proposed by Kipf and Welling [39], where N denotes the number of nodes in the graph. All the units are active irrespective of the dimension of latent space. In case of EVGAE, the number of active units is in between the two. i.e. we are able to mitigate the over-pruning without sacrificing the generative ability ($\beta = 1$). This results in better performance in graph analysis tasks as shown in table 2.1.

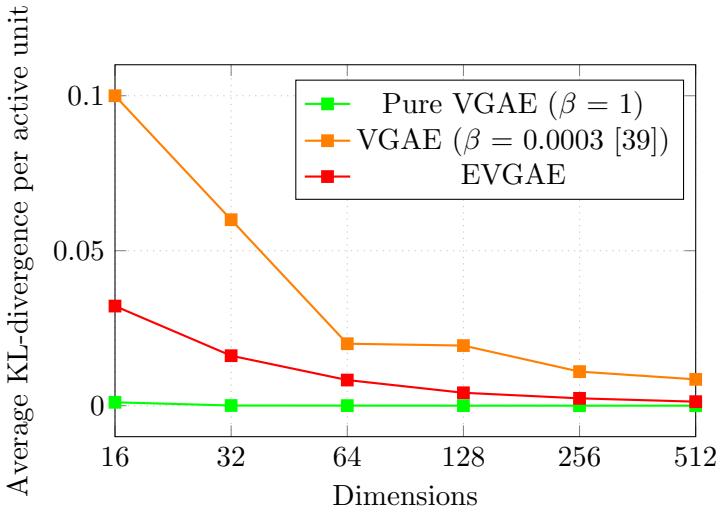
To demonstrate that EVGAE achieves better distribution matching than VGAE, we compare the average KL-divergence of active units for different latent space dimensions. Only active units are considered when averaging the KL-divergence because the inactive units introduce a bias towards zero in the results. Figure 2.6(b) shows how the distribution matching varies as we increase the number of dimensions. We note that when $\beta = 1$, the average KL-divergence for active unit is still quite small, indicating a good match between learned latent distribution and the prior. Conversely, when $\beta = N^{-1}$ the average KL-divergence per active unit is quite high. This supports our claim that original VGAE [39] learns a latent distribution which is quite different from the prior. Thus, when we generate new samples from standard gaussian distribution and pass it through the decoder, we get quite different output than the graph data used for training. In case of EVGAE, the KL divergence is quite closer to the prior compared to VGAE. For $D = 512$, it is almost similar to the case with $\beta = 1$.

2.6 Conclusion

In this paper we looked at the issue of over-pruning in variational graph autoencoder. We demonstrated that the way VGAE [39] deals with this issue results in a latent distribution which is quite different from the standard gaussian prior. We proposed an alternative model based approach EVGAE that mitigates the problem of over-pruning by encouraging more latent variables to actively play their role in the reconstruction. EVGAE also has a better generative ability than VGAE [39] i.e. better matching between learned and prior distribution. Moreover, EVGAE performs comparable or slightly better than the popular methods for the link prediction task.



(a) Active hidden units with varying latent space dimensions



(b)

Figure 2.6: Effect of changing latent space dimensions on active units and their KL-divergence. It can be observed that EVGAE has more active units compared to VGAE, and with better generative ability

3 Barlow Graph Autoencoder

Based on the following publication:

Khan, R.A.* *and Kleinsteuber, M., 2021. Barlow Graph Auto-Encoder for Unsupervised Network Embedding. arXiv preprint arXiv:2110.15742. Manuscript under-review at AISATS 2023*

Over the years, a variety of approaches have been proposed for learning network embedding. On one hand, we have techniques that aim to learn the network embedding by employing the proximity information which is not limited to the first-hop neighbors. Such approaches include spectral, random-walk based and matrix-factorization-based methods [63, 75, 78, 79]. On the other hand, most neural network-based approaches focus on the structural information by limiting themselves to the immediate neighborhood [40, 46, 47]. Intuitively, a larger neighborhood offers richer information that should consequently help in learning better network embedding. However, the neural network-based approaches often yield better results compared to the spectral techniques etc., despite them being theoretically more elegant. Recently, graph Diffusion [80] has been proposed to enable a variety of graph-based algorithms to make use of a larger neighborhood in the graphs with high homophily. This is achieved by precomputing a graph diffusion matrix from the adjacency matrix and then using it in place of the original adjacency matrix. For instance, coupling this technique with graph neural networks (GNNs) enables them to learn from a larger neighborhood, thereby improving network embedding learning. However, replacing the adjacency matrix with the diffusion matrix deprives the algorithms of an explicit local view provided by the immediate neighborhood, and forces them to learn only from the global view presented by the diffusion matrix. Such an approach can affect the performance of the learning algorithm, especially in the graphs where the immediate neighborhood holds high significance. This advocates the need to revisit how the information in the multi-hop neighborhood is employed to learn the network embedding. There exist some contrastive approaches like [49–51] that can capture the information in a larger neighborhood in the form of the summary vectors, and then learn network embedding by aiming to maximize the local-global mutual information between local node representations and the global summary vectors. However, this information is captured in an *implicit* manner in the sense that there is no objective function to ensure the preservation of the information in the larger neighborhood.

In this work, we adopt a novel approach for learning network embedding by simultaneously employing the information in the immediate as well as the larger neighborhood in an *explicit* manner. This is achieved by learning concurrently from the adjacency matrix and the graph diffusion matrix. To efficiently merge the two sources of information, we take inspiration from Barlow Twins, an approach recently proposed for unsupervised learning of the image embeddings by constructing the cross-correlation matrix between the outputs of two identical networks fed with distorted versions of image samples, and making it as close to the identity matrix as possible [53]. Motivated by this, we propose an autoencoder-based architecture named **B**arlow **G**raph **A**uto-**E**ncoder(BGAE), along with its variational counterpart named **B**arlow **V**ariational **G**raph **A**uto-**E**ncoder(BVGAE). Both BGAE and BVGAE make use of the immediate as well as the larger neighborhood information to learn network embedding in an unsupervised manner while minimizing the redundancy between the components of the low-dimensional projections. Our contribution is three-fold:

- We propose a simple yet effective autoencoder-based architecture for unsupervised network embedding, which *explicitly* learns from both the immediate and the larger neighborhoods provided in the form of the adjacency matrix and the graph diffusion matrix respectively.
- Motivated by Barlow Twins, BGAE and BVGAE aim to achieve stability towards distortions and redundancy-minimization between the components of the embedding vectors.
- We show the efficacy of our approach by evaluating it on link prediction, transductive node classification and clustering on eight benchmark datasets. Our approach consistently yields promising results for all the tasks whereas the included competitors often under-perform on one or more tasks.

As detailed in the subsequent sections and derivations, it is non-trivial to efficiently merge the information from neighborhoods at different levels, as it needs a careful choice of the loss function and related architecture components.

3.1 Related Work

3.1.1 Network Embedding

Earlier work related to network embedding, such as GraRep [81], HOPE [82], and M-NMF [79], etc., employed matrix factorization-based techniques. Concurrently, some probabilistic models were proposed to learn network embedding by using random-walk-based objectives. Examples of such approaches include DeepWalk [63], Node2Vec [75], and LINE [83], etc. Such techniques over-emphasize the information in proximity, thereby sacrificing the structural information [84]. In recent years

several graph neural network (GNN) architectures have been proposed as an alternative to matrix-factorization and random-walk-based methods for learning graph-domain tasks. Some well-known examples of such architectures include graph convolutional network or GCN [40], graph attention network or GAT [46], Graph Isomorphism Networks or GIN [47], and GraphSAGE [48], etc. This has allowed exploration of network embedding using GNNs [25, 26]. Such approaches include autoencoder based (e.g., VGAE [39] and GALA [85]), adversarial (e.g, ARVGA [86] and DBGANp [87]), and contrastive techniques (e.g., DGI [49], MVGRL [88] and GRACE [51]), etc.

3.1.2 Barlow Twins

This approach has been recently proposed [53] as a self-supervised learning (SSL) mechanism making use of redundancy reduction - a principle first proposed in neuroscience [52]. Barlow Twins employs two identical networks, fed with two different versions of the batch samples, to construct two versions of the low-dimensional projections. Afterward, it attempts to equate the cross-correlation matrix computed from the twin projections to identity, hence reducing the redundancy between different components of the projections. This approach is relatable to several well-known objective functions for SSL, such as the information bottleneck objective [89], or the INFONCE objective [90].

The idea of Barlow Twins has been ported recently to graph datasets by Graph Barlow Twins or G-BT [91]. Inspired by the image augmentations proposed by Barlow Twins (cropping, color jittering, blurring, etc.), G-BT adopts edge dropping and node feature masking to form the augmented views of the input graphs. While this approach works for transductive node classification, its performance degrades for tasks involving link prediction as demonstrated by the experiments in section 3.3 because there is no explicit objective to preserve the information in links. As we will see in section 3.2.2.2, the addition of this objective is non-trivial because it involves a careful modification of the original loss term from Barlow Twins.

3.1.3 Graph Diffusion Convolution (GDC)

GDC [80] was proposed as a way to efficiently aggregate information from a large neighborhood. This is achieved in two steps.

1. **Diffusion:** First a dense diffusion matrix $\overline{\mathbf{S}}$ is constructed from the adjacency matrix using generalized graph diffusion as a denoising filter.

2. **Sparsification:** This is the second step where either top k entries of $\overline{\mathbf{S}}$ are selected in every row, or the entries below a threshold ϵ are set to 0. We use $\mathbf{S}$ to denote the resulting sparse diffused matrix. The value of the threshold can also be estimated from the intended average degree of the sparse graph [80].

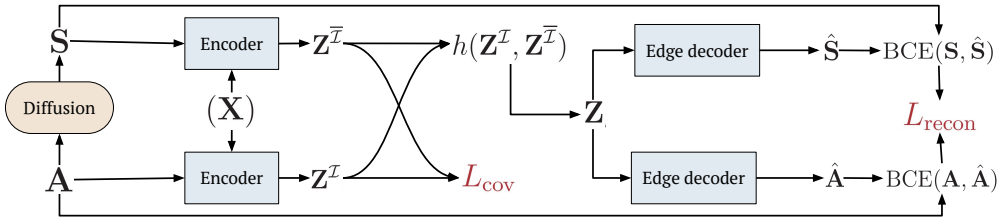


Figure 3.1: General model architecture. Based upon the input adjacency matrix $\mathbf{A}$, we get the diffusion matrix $\mathbf{S}$, which needs to be computed only once. The encoder yields the low dimensional projection matrices $\mathbf{Z}^{\mathcal{I}}$ and $\mathbf{Z}^{\overline{\mathcal{I}}}$ corresponding to $\mathcal{I}$ and $\overline{\mathcal{I}}$ respectively. Using a fusion function $h(\cdot)$, The projections $\mathbf{Z}^{\mathcal{I}}$ and $\mathbf{Z}^{\overline{\mathcal{I}}}$ are fused into a single projection $\mathbf{Z}$, which is then fed to the edge decoder to reconstruct $\hat{\mathbf{A}}$ and $\hat{\mathbf{S}}$.

$\mathbf{S}$ defines an alternate graph with weighted edges that carry more information than a binary adjacency matrix. This sparse matrix, when used in place of the original adjacency matrix, improves graph learning for a variety of graph-based models such as degree-corrected stochastic block model or DCSBM [92], DeepWalk, GCN, GAT, GIN, and DGI, etc.

3.2 Barlow Graph AutoEncoder

3.2.1 Problem Formulation

Suppose an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the adjacency matrix $\mathbf{A} \in \{0, 1\}^{N \times N}$ and optionally a matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ of F-dimensional node features, N being the number of nodes. In addition, we construct a *diffused* version of $\mathcal{G}$ by building a diffusion matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$ from $\mathbf{A}$. For brevity we use $\mathcal{I} = (\mathbf{A}, \mathbf{X})$ and $\overline{\mathcal{I}} = (\mathbf{S}, \mathbf{X})$ if features are available, otherwise $\mathcal{I} = \mathbf{A}$ and $\overline{\mathcal{I}} = \mathbf{S}$. Given d as the embedding-dimension size, we aim to optimize the model parameters for finding the network embeddings $\mathbf{Z}^{\mathcal{I}} \in \mathbb{R}^{N \times d}$ and $\mathbf{Z}^{\overline{\mathcal{I}}} \in \mathbb{R}^{N \times d}$ from $\mathcal{I}$ and $\overline{\mathcal{I}}$ such that:

1. $\mathbf{Z}^{\mathcal{I}}$ and $\mathbf{Z}^{\overline{\mathcal{I}}}$ can be fused in a way that both $\mathbf{A}$ and $\mathbf{S}$ can be reconstructed from the fused embedding $\mathbf{Z} \in \mathbb{R}^{N \times d}$. This allows the embedding to capture the local information from $\mathcal{I}$ as well as the information in a larger neighborhood from $\overline{\mathcal{I}}$.

2. Same components of different projections have high covariance and vice versa for different components. This adds to the stability towards distortions and also reduces redundancy between different components of the embedding.

Mathematically, the above objective is achieved by minimizing the following loss function

$$L = L_{\text{recon}} + \beta L_{\text{cov}} \tag{3.1}$$

where β is a hyperparameter of the algorithm to weigh between the reconstruction loss and the cross-covariance loss. The general model architecture is given in figure 3.1. We first define the loss terms and then brief the modules of the architecture.

3.2.2 Loss Terms

3.2.2.1 Reconstruction Loss (L_{recon})

We aim to learn the free model parameters θ in order to maximize the log probability of recovering both $\mathbf{A}$ and $\mathbf{S}$ from $\mathbf{Z}$. This probability can be written as a marginalization over the joint distribution containing the latent variables $\mathbf{Z}$ as

$$\log\Big(p_\theta(\mathbf{A},\mathbf{S})\Big) = \log\Big(\int p_\theta(\mathbf{A},\mathbf{S},\mathbf{Z})d\mathbf{Z}\Big) \tag{3.2}$$

$$= \log\Big(\int p(\mathbf{Z})p_\theta(\mathbf{A}|\mathbf{Z})p_\theta(\mathbf{S}|\mathbf{Z})d\mathbf{Z}\Big), \tag{3.3}$$

where the prior $p(\mathbf{Z})$ is modelled as a unit gaussian. Equation 3.3 assumes conditional independence between $\mathbf{A}$ and $\mathbf{S}$ given $\mathbf{Z}$.

To ensure tractability, we introduce the approximate posterior $q_\phi(\mathbf{Z}|\mathcal{I},\overline{\mathcal{I}})$, parameterized by ϕ as

$$q_\phi(\mathbf{Z}|\mathcal{I},\overline{\mathcal{I}}) = q_\phi(\mathbf{Z}^{\mathcal{I}}\mathbf{Z}^{\overline{\mathcal{I}}}|\mathcal{I},\overline{\mathcal{I}}) \tag{3.4}$$

$$= q_\phi(\mathbf{Z}^{\mathcal{I}}|\mathcal{I})q_\phi(\mathbf{Z}^{\overline{\mathcal{I}}}|\overline{\mathcal{I}}), \tag{3.5}$$

where equation 3.5 follows from equation 3.4 because of the assumed conditional independence of $\mathbf{Z}^{\mathcal{I}}$ and $\mathbf{Z}^{\overline{\mathcal{I}}}$ given their respective inputs. Both $q_\phi(\mathbf{Z}^{\mathcal{I}}|\mathcal{I})$ and $q_\phi(\mathbf{Z}^{\overline{\mathcal{I}}}|\overline{\mathcal{I}})$ are modelled as Gaussians by a single encoder block that learns the parameters $\mu(\cdot)$ and $\sigma(\cdot)$ of the distribution conditioned on the given inputs $\mathcal{I}$ and $\overline{\mathcal{I}}$ respectively. The term L_{recon} can now be considered as a negative of the ELBO bound derived as

$$\begin{aligned}\log\big(p_\theta(\mathbf{A},\mathbf{S})\big) \geq &- D_{KL}\Big(q_\phi(\mathbf{Z}^{\mathcal{I}}|\mathcal{I}) \;||\; \mathcal{N}(\mathbf{0},\mathbf{I})\Big) \\ &- D_{KL}\Big(q_\phi(\mathbf{Z}^{\overline{\mathcal{I}}}|\overline{\mathcal{I}}) \;||\; \mathcal{N}(\mathbf{0},\mathbf{I})\Big) \\ &- \text{BCE}\big(\hat{\mathbf{A}},\mathbf{A}\big) - \text{BCE}\big(\hat{\mathbf{S}},\mathbf{S}\big) \qquad (3.6)\\ =&\mathcal{L}_{\text{ELBO}} \qquad (3.7)\\ =&- L_{\text{recon}}, \qquad (3.8)\end{aligned}$$

where D_{KL} refers to the KL divergence, BCE is the binary cross-entropy, and the matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{S}}$ refer to reconstructed versions of $\mathbf{A}$ and $\mathbf{S}$ respectively. The inequality in 3.6 follows from Jensen's inequality. It is worth noticing that BCE

is computed based on the edges constructed from the fused embedding $\mathbf{Z}$. For detailed derivation, we refer the reader to the supplementary material.

The reconstruction loss in 3.8 refers to the variational variant BVGAE. For the non-variational case of BGAE, the KL divergence terms get dropped, leaving only the two BCE terms.

3.2.2.2 Covariance Loss (L_{cov})

The correlation-loss in [53] (also employed in [91]) involves normalization by the standard deviation of the embedding vectors, centered across the input batch (equation 2 in [53]). While this works for images and for nodes of the graphs, it has a tendency to obscure the information in the relative strengths of the links in a graph. For graph datasets, we often replace cosine similarity with dot products, followed by a sigmoid (as done in [39], [93], and [88], etc.) as it helps in preventing the information in the magnitude of the vectors. Following this approach, instead of computing the cross-correlation matrix, we compute the cross-covariance matrix and use the sigmoid function to individually normalize the absolute entries $c_{\ell m}$. The loss L_{cov} is then computed as a summation of two terms corresponding to the mean of diagonal elements and off-diagonal elements of $\mathcal{C}$.

$$L_{\text{cov}} = -\frac{1}{N}\sum_{\ell=m}\log(c_{mm}) - \frac{\lambda}{N(N-1)}\sum_{\ell\neq m}\log(1-c_{\ell m}), \tag{3.9}$$

where λ defines the trade-off between the two terms. The first term of equation 3.9 is the invariance term. When minimized, it makes the embedding stable towards distortions. The second term refers to the cross-covariance between different components of $\mathcal{C}$. When minimized, it reduces the redundancy between different components of the vectors. The entries $c_{\ell m}$ of $\mathcal{C}$ are given by

$$c_{\ell m} = \text{sigmoid}\Big(\big|\sum_{b=1}^{|\mathcal{B}|}(z^{\mathcal{I}}_{b\ell} - \overline{z}^{\mathcal{I}}_{\ell})(z^{\overline{\mathcal{I}}}_{bm} - \overline{z}^{\overline{\mathcal{I}}}_{m})\big|\Big), \tag{3.10}$$

where b indexes the batch $\mathcal{B}$ with size $|\mathcal{B}|$ and the ℓ-th component of the latent embedding $\mathbf{z}^{\mathcal{I}}_b$ is denoted by $z^{\mathcal{I}}_{b\ell}$. The empirical means across embeddings $\mathbf{Z}^{\mathcal{I}}$ and $\mathbf{Z}^{\overline{\mathcal{I}}}$ are denoted by $\overline{z}^{\mathcal{I}}$ and $\overline{z}^{\overline{\mathcal{I}}}$ respectively. It is also worth noticing that the underlying objective is the same for equation 3.9 as well as the original correlation-based loss in Barlow Twins i.e., $\mathcal{C}$ should be as close to the identity matrix as possible.

3.2.3 Model Architecture Blocks

We now describe the modules leading to the loss terms in equation 3.1 as shown in figure 3.1.

3.2.3.1 Diffusion:

The generalized diffusion to construct $\mathbf{S}$ from $\mathbf{A}$ is given by

$$\mathbf{S} = \sum_{k=0}^{\infty} \theta_k \mathbf{T}^k, \tag{3.11}$$

where $\mathbf{T}$ is the generalized transition matrix and θ_k are the weighting coefficients. There can be multiple possibilities for θ_k and $\mathbf{T}^k$ while ensuring the convergence of equation 3.11 such as the ones proposed in [94], [95], and [80] etc. In this work, we report the case of Personalized PageRank (PPR: [96]) as it consistently gave better results with BGAE/BVGAE. For the detailed results including the Heat Kernel [80], we refer the reader to the supplementary material.

PPR kernel corresponds to $\mathbf{T} = \mathbf{A}\mathbf{D}^{-1}$ and $\theta_k = \alpha(1-\alpha)^k$, where $\mathbf{D}$ is the degree matrix and $\alpha \in (0, 1)$ is the teleport probability. The corresponding symmetric transition matrix is given by $\mathbf{T} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. Substitution of $\mathbf{T}$ and θ_k into equation 3.11 leads to a closed form solution for diffusion using PPR kernel as

$$\mathbf{S} = \alpha\Big(\mathbf{I} - (1-\alpha)\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\Big)^{-1}. \tag{3.12}$$

Equation 3.12 restricts the use of PPR-based diffusion for large graphs. However, in practice, there exist multiple approaches to efficiently approximate equation 3.12 such as the ones proposed by [97] and [98]. In all the results reported in this paper, we use the approximate version of equation 3.12. Diffusion works well for graphs with high homophily. So BGAE/BVGAE also target similar networks.

3.2.3.2 Encoder:

This module is responsible for projecting the information in $\mathcal{I}$ and $\overline{\mathcal{I}}$ into d-dimensional embeddings $\mathbf{Z}^{\mathcal{I}}$ and $\mathbf{Z}^{\overline{\mathcal{I}}}$ respectively. Following [53], we use a single encoder to encode both versions of the input graph. Our framework is general in the sense that any reasonable encoder can be plugged in to get the learnable projections.

In our work, we have considered two options for the encoder block, consequently leading to two variants of the framework.

- For BGAE, we use a single-layer GCN encoder.
- For BVGAE, we employ a simple variational encoder that learns the parameters $\mu(\cdot)$ and $\sigma(\cdot)$ of a gaussian distribution conditioned upon the input samples. The latent samples can then be generated by following the reparameterization trick [19].

3.2.3.3 Fusion Function:

The function $h(\cdot)$ is used to fuse $\mathbf{Z}^{\mathcal{I}}$ and $\mathbf{Z}^{\overline{\mathcal{I}}}$ into a single matrix $\mathbf{Z}$. In this work we define $\mathbf{Z}$ as a weighted sum of $\mathbf{Z}^{\mathcal{I}}$ and $\mathbf{Z}^{\overline{\mathcal{I}}}$ as

$$h(\mathbf{z}_i^{\mathcal{I}}, \mathbf{z}_i^{\overline{\mathcal{I}}}) = \psi_i^{\mathcal{I}} \mathbf{z}_i^{\mathcal{I}} + \psi_i^{\overline{\mathcal{I}}} \mathbf{z}_i^{\overline{\mathcal{I}}}, \tag{3.13}$$

where the weights $\{\psi_i^{\mathcal{I}}\}_{i=1}^N$ and $\{\psi_i^{\overline{\mathcal{I}}}\}_{i=1}^N$ can either be fixed or learned. In this work, we report two variants of the fusion function.

- Fixing $\psi_i^{\mathcal{I}} = \psi_i^{\overline{\mathcal{I}}} = 0.5 \; \forall i$.
- Learning $\{\psi_i^{\mathcal{I}}\}_{i=1}^N$ and $\{\psi_i^{\overline{\mathcal{I}}}\}_{i=1}^N$ using attention mechanism.

For attention, we compute the dot products of different embeddings of the same node with respective learnable weight vectors, followed by LeakyReLU activation. Afterward, a softmax is applied to get the probabilistic weight assignments, i.e.

$$\psi_i^{\mathcal{I}} = \frac{\exp\left(\mathrm{A}\left(\mathbf{w}_1^T \mathbf{z}_i^{\mathcal{I}}\right)\right)}{\exp\left(\mathrm{A}\left(\mathbf{w}_1^T \mathbf{z}_i^{\mathcal{I}}\right)\right) + \exp\left(\mathrm{A}\left(\mathbf{w}_2^T \mathbf{z}_i^{\overline{\mathcal{I}}}\right)\right)}, \tag{3.14}$$

$$\psi_i^{\overline{\mathcal{I}}} = 1 - \psi_i^{\mathcal{I}}. \tag{3.15}$$

where $\mathrm{A}(\cdot)$ is the LeakyReLU activation function and $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$ are learnable weight vectors.

3.2.3.4 Decoder

Since our approach is autoencoder-based, we use the edge decoder as proposed by [39], to reconstruct the entries $\hat{\mathrm{a}}_{ij}$ of $\hat{\mathbf{A}}$ as

$$\hat{\mathrm{a}}_{ij} = \mathrm{sigmoid}(\mathbf{z}_i^T \mathbf{z}_j) \tag{3.16}$$

The entries of $\hat{\mathbf{S}}$ can also be reconstructed similarly.

3.3 Experiments

This section describes the datasets and the experiments conducted to evaluate the efficacy of our approach. We choose eight benchmark datasets including Wikipedia articles (WikiCS) [99], Amazon co-purchase data networks (AmazonPhoto and AmazonComputers [100]), extracts from Microsoft Academic Graph (CoauthorCS and CoauthorPhysics) [101], and citation networks (Cora, CiteSeer and PubMed) [68]. The basic characteristics of these datasets are briefed in table 3.1. We first report the results for link prediction. The network embedding learned by BGAE and

Dataset	Nodes	Edges	Features	Classes
Cora [68]	2,708	5,297	1,433	7
CiteSeer [68]	3,312	4,732	3,703	6
PubMed [68]	19,717	44,338	500	3
WikiCS [99]	11,701	216,123	300	10
AmazonComputers [100]	13,752	245,861	767	10
AmazonPhotos [100]	7,650	119,081	745	8
CoauthorCS [101]	18,333	81,894	6,805	15
CoauthorPhysics [101]	34,493	247,962	8,415	5

Table 3.1: Datasets used for evaluation.

BVGAE is unsupervised as no node labels are used during training. Hence, to measure the quality of the network embedding, we analyze our approach for two downstream tasks: clustering and transductive node classification. For interested readers, the supplementary material contains a detailed analysis of BGAE and BVGAE in different settings. We use the AWS EC2 instance type `g4dn.4xlarge` with 16GB GPU for training. For reproducibility, the implementation details of all the experiments along with the code are provided in [102]. For all the experiments, we report publicly available results from our competitors.

3.3.1 Link Prediction

3.3.1.1 Comparison Methods

For link prediction, we select 12 competitors. We start with **DeepWalk** [63] as the baseline.

Autoencoder based Architectures: We include graph autoencoder or **GAE** which aims to reconstruct the adjacency matrix for the input graph. The variational graph autoencoder or **VGAE** [39] is its variational counterpart that extends the idea of variational autoencoder or VAE [19] to the graph domain. In the case of adversarially regularized graph autoencoder or **ARGA** [86], the latent representation is forced to match the prior via an adversarial training scheme. Just like VGAE, there exists a variational alternative to ARGA, known as adversarially regularized variational graph autoencoder or **ARVGA**. **GALA** [85] learns network embedding by treating encoding and decoding steps as Laplacian smoothing and Laplace sharpening respectively.

Contrastive Methods: **DGI** [49] leverages Deep-Infomax [103] for graph datasets. Graph InfoClust or **GIC** [104] learns network embedding by maximizing the mutual information with respect to the graph-level summary as well as the cluster-level summaries. **GMI** [105] aims to learn node representations while aiming to improve generalization performance via added contrastive regularization. **GCA** [106] proposes adaptive augmentation techniques to contrast views between

Algorithm	Cora		CiteSeer		PubMed		CoauthorCS		AmazonPhoto	
	AUC	**AP**	**AUC**	**AP**	**AUC**	**AP**	**AUC**	**AP**	**AUC**	**AP**
DeepWalk	83.10	85.00	80.50	83.60	84.40	84.10	91.74	91.19	91.48	90.79
GAE	91.00	92.00	89.50	89.90	96.40	96.50	94.09	93.86	93.86	92.96
VGAE	91.40	92.60	90.80	92.00	94.40	94.70	89.60	89.36	92.05	92.02
ARGA	92.40	93.20	91.90	93.00	96.80	97.10	91.99	92.54	**96.10**	**95.40**
ARVGA	92.40	92.60	92.40	93.00	96.50	96.80	93.32	93.32	92.70	90.90
GALA	92.10	92.20	94.40	94.80	91.50	89.70	93.81	94.49	91.80	91.00
DGI	89.80	89.70	95.50	95.70	91.20	92.20	94.87	94.34	92.24	92.14
GIC	93.50	93.30	97.00	96.80	93.70	93.50	95.03	94.94	92.70	92.34
GMI	95.10	95.60	97.80	97.40	96.37	96.04	<u>96.37</u>	95.04	93.88	92.67
GCA	95.75	95.47	96.44	96.49	95.28	95.52	96.31	**96.28**	93.25	92.74
MVGRL	90.52	90.45	92.89	92.89	92.45	92.17	95.17	95.58	92.89	92.45
G-BT	87.46	86.84	93.42	93.01	94.53	94.26	92.64	91.40	95.12	95.45
BGAE	<u>99.02</u>	<u>98.82</u>	<u>98.59</u>	**98.61**	97.78	97.68	96.31	95.44	95.01	94.24
BGAE + Att	**99.35**	**99.23**	98.56	<u>98.57</u>	**98.06**	**98.03**	**96.51**	<u>95.65</u>	<u>95.18</u>	<u>94.49</u>
BVGAE	97.87	97.62	**98.63**	<u>98.57</u>	<u>97.93</u>	<u>97.89</u>	96.12	95.13	94.61	94.29
BVGAE + Att	98.03	97.77	98.23	98.08	97.77	97.74	96.21	95.34	94.97	94.28

Table 3.2: Link prediction performance, as evaluated by AUC and AP metrics. The best results are styled as bold and the second best are underlined.

nodes and subgraphs or structurally transformed graphs. **MVGRL** [88] learns graph embedding by contrasting multiple views of the input data.

In addition, we include **G-BT** [91] which is another approach making use of the redundancy-minimization principle introduced in [53] as discussed in section 3.1.2.

For link prediction, we skip the three datasets where many public results are missing and report these results in the supplementary material.

3.3.1.2 Settings

For link prediction, we follow the same link split as adopted by our competitors, i.e., we split the edges into the training, validation, and test sets containing 85%, 5%, and 10% links respectively. For all the competitors we keep the same settings as given by the authors. For the methods where multiple variants are given by the authors(e.g., ARGA, ARVGA, etc.), we report the best results among all the variants. For our approach, we keep the latent dimension to 512 for all the experiments except CoauthorPhysics where $d = 128$ to avoid out-of-memory issues. We use a single layer GCN encoder for BGAE, and two GCN encoders to output the parameters $\mu(\cdot)$ and $\sigma(\cdot)$ in case of the variational encoder block of BVGAE. For all the experiments, we get $\mathbf{S}$ by setting the average degree to 25. The value of the hyperparameter λ in equation 3.9 is set to $5e^{-3}$ for all the experiments. This is the same as proposed in Barlow-Twins [53]. Adam [76] is used as the optimizer with learning rate and rate decay set to 0.01 and $5e^{-6}$. The hyperparameter β is fixed to 1 for all experiments. Instead of computing a closed-form solution, it is sufficient to compute the BCE loss using the samples $\mathbf{z}$. For this, we follow other autoencoder-based approaches such as [39, 93] for dataset splits and sampling of positive/negative edges for every training iteration. For evaluation, we report the area under the curve (AUC) and average precision (AP) metrics. All the results are the average of 10 runs. Further implementation details can be found in [102].

3.3.1.3 Results

Table 3.2 gives the results for the link prediction task. For our approach, we give the results for BGAE as well as BVGAE, both with and without attention. We can observe that our approach achieves the best or second-best results in all the datasets. Overall the variant of BGAE with attention performs well across all the datasets and metrics. This validates the choice of attention as a fusion function. Among the competitors, the contrastive approaches perform relatively better across all the datasets. One exception is AmazonPhoto where ARGA achieves the best results. However, as the next sections demonstrate, the performance of ARGA/ARVGA degrades for downstream node classification and clustering. Another thing to note is the results of G-BT. As the training epochs go on for G-BT, the results degrade rapidly, often by about 30% of the results reported in table 3.2. Apart from AmazonPhotos, there is a healthy margin between BGAE and G-BT mainly

Algorithm	**Cora**	**CiteSeer**	**PubMed**	**WikiCS**	**CoauthorCS**	**CoauthorPhysics**	**AmazonComputers**	**AmazonPhoto**
Raw	47.87	49.33	69.11	71.98	90.37	93.58	73.81	78.53
DeepWalk	70.66	51.39	74.31	77.21	87.70	94.90	86.28	90.05
GAE	71.53	65.77	72.14	70.15	90.01	94.92	85.18	91.68
VGAE	75.24	69.05	75.29	75.63	92.11	94.52	86.44	92.24
ARGA	74.14	64.14	74.12	66.88	89.41	93.10	84.39	<u>92.68</u>
ARVGA	74.38	64.24	74.69	67.37	88.54	94.30	84.66	92.49
DGI	81.68	71.47	77.27	75.35	92.15	94.51	83.95	91.61
GIC	81.73	71.93	77.33	77.28	89.40	93.10	84.89	92.11
GRACE	80.04	71.68	79.53	**80.14**	92.51	94.70	87.46	92.15
GMI	83.05	<u>73.03</u>	80.10	74.85	OOM	OOM	82.21	90.68
GCA	82.10	71.30	80.20	78.23	92.95	**95.73**	88.94	92.53
MVGRL	82.90	72.60	79.40	77.52	92.11	92.11	87.52	91.74
BGRL	82.70	71.10	79.60	<u>79.98</u>	93.31	95.56	89.68	**92.87**
G-BT	80.80	73.00	80.00	76.65	92.95	95.07	88.14	92.63
BGAE	<u>83.51</u>	72.43	**81.84**	78.93	**93.76**	95.01	<u>92.24</u>	91.10
BGAE + Att	**83.60**	72.41	<u>80.95</u>	79.53	**93.76**	<u>95.64</u>	**92.44**	91.89
BVGAE	82.62	72.97	80.02	77.52	93.25	95.13	89.19	89.38
BVGAE + Att	82.57	**73.09**	80.25	77.82	93.15	95.60	89.91	89.98

Table 3.3: Transductive node classification performance, as evaluated by accuracy. The best results are styled as bold and second best are underlined.

because unlike BGAE, G-BT does not explicitly preserve the information in the links.

3.3.2 Transductive Node Classification

3.3.2.1 Comparison Methods

We compare with 14 competitors for transductive node classification, using raw features and DeepWalk(with features) as the baselines. In addition to the methods briefed in section 3.3.1.1, we include two more contrastive methods i.e. GRACE and BGRL: **GRACE** [51] learns network embedding by making use of multiple views and contrasting the representation of a node with its raw information (e.g., node features) or neighbors' representations in different views. **BGRL** [107] eliminates the need for negative samples by minimizing invariance between two augmented versions of mini-batches of graphs.

3.3.2.2 Settings

The training phase uses the same settings as reported in section 3.3.1.2. For transductive node classification, we do not need to split the edges into training/-validation/test sets. So we use all the edges for self-supervised learning of the node embeddings. For evaluating the embedding, a logistic regression head is used with `lbfgs` solver. For this, we use the default settings of the `scikit-learn` package. For citation datasets (Cora, CiteSeer, and PubMed), we follow the standard public splits for training/validation/test sets used in many previous works such as [40, 49, 108, 109], i.e., 20 labels per class for training, 500 samples for validation, and 1000 for testing. For WikiCS, we average over the 20 splits that are publicly provided. For the rest of the datasets (AmazonPhoto, AmazonComputers, CoauthorPhysics, and coauthorCS), we follow the split configuration of B-JT, i.e. generate random splits with training, validation, and test sets containing 10%, 10%, and 80% nodes respectively. For evaluation, we use accuracy as the metric.

3.3.2.3 Results

Table 3.3 gives the comparison between different algorithms for transductive node classification. We can again observe consistently good results with our approach for all eight datasets. For this task, the margin is rather small, especially for Cora and CiteSeer, compared to the best competitor i.e. GMI. Nonetheless, our point still stands well-conveyed that our approach performs on par with the well-known network embedding techniques for transductive node classification. A comparison of table 3.3 with table 3.2 demonstrates inconsistencies in the performance of our competitors for the two tasks. This is mainly because either the competitors do not explicitly preserve information in the links (e.g. MVGRL, G-BT, etc), or link prediction is their main focus (e.g., in GAE/ARGA, etc). For instance, ARGA

Algorithm	Cora	CiteSeer	PubMed	WikiCS	CoauthorCS	CoauthorPhysics	AmazonComputers	Amazon-Photos
K-means	32.10	30.50	0.10	18.20	64.20	48.90	16.60	28.20
GAE	42.90	17.60	27.70	24.30	73.10	54.50	44.10	61.60
VGAE	43.60	15.60	22.90	26.10	73.30	56.30	42.30	53.00
ARGA	44.90	35.00	30.50	27.50	66.80	51.20	23.50	57.70
ARVGA	52.60	33.80	29.00	28.70	61.60	52.60	23.70	45.50
DGI	41.10	31.50	27.70	31.00	74.70	67.00	31.80	37.60
GRACE	46.18	38.29	16.27	42.82	75.62	OOM	47.93	65.13
GCA	55.70	37.40	28.90	29.90	73.50	59.40	42.60	34.40
MVGRL	60.90	**44.00**	31.50	26.30	74.00	59.40	24.40	34.40
G-BT	43.40	41.57	29.52	27.46	74.37	59.8	65.55	52.39
BGAE	**62.42**	43.36	<u>38.46</u>	<u>45.80</u>	<u>80.10</u>	<u>68.01</u>	**66.98**	<u>67.13</u>
BGAE + Att	<u>62.27</u>	<u>43.84</u>	**38.59**	**46.93**	**80.30**	**68.12**	<u>66.93</u>	**67.43**
BVGAE	59.60	43.29	37.41	40.78	79.01	67.10	60.98	61.33
BVGAE + Att	59.82	43.27	37.47	40.86	79.42	67.06	61.44	61.62

Table 3.4: Node clustering performance, as evaluated by NMI. The best results are styled as bold and second best are underlined. OOM refers to Out-of-Memory.

performed reasonably well for link prediction but fails to give a similar consistent performance across all datasets in table 3.3. On the other hand, MVGRL performs well in table 3.3, although it suffered in table 3.2.

3.3.3 Node Clustering

3.3.3.1 Settings

For clustering, we choose 10 methods in total for comparison, with the baseline established by K-Means. The experimental configuration for node clustering follows the same pattern as in section 3.3.1.2. For clustering, we use all the edges just like in section 3.3.2.2, i.e., all the edges are used for self-supervised learning of the node embeddings. Afterward, we use K-Means to infer the cluster assignments from the embeddings. For evaluation, we use normalized mutual information (NMI) as the metric.

3.3.3.2 Results

The results of the experiments for downstream node clustering are given in table 3.4. Here again, we perform consistently well for all the datasets except CiteSeer, where we achieve the second-best results by a small margin. It is worth noticing that apart from CiteSeer, we achieve both the best and the second results using different variants of BGAE. A comparison of table 3.3 with table 3.2 and table 3.4 again highlights that no competitor algorithm performs consistently well for all the tasks and datasets. For instance, ARGA performed well on some datasets in table 3.2. However, its performance suffers in table 3.4. Similarly, GMI, which performs well in table 3.3, is outperformed by many other algorithms in node clustering. On the other hand, the algorithms such as GIC, that perform well in table 3.4 are outperformed by others in table 3.3. This highlights the task-specific nature of the network embedding learned by different competitors and also shows the efficacy of BGAE across multiple tasks and datasets.

3.4 Conclusion

This work proposes a simple yet effective autoencoder-based approach for network embedding that simultaneously employs the information in the immediate and larger neighborhoods. To construct a uniform network embedding, the two information sources are efficiently coupled using the redundancy-minimization principle. We propose two variants, BGAE and BVGAE, depending upon the type of encoder block. To construct a larger neighborhood from the immediate neighborhood, we use graph diffusion. Our work is restricted to the networks with high homophily because diffusion only works well for such networks. As demonstrated by the extensive experimentation, our approach is on par with the well-known baselines,

often outperforming them over a variety of tasks such as link prediction, clustering, and transductive node classification.

Part II

Community-Aware Network Embedding

4 Case of Homogeneous Networks

Based on the following peer-reviewed publication:

Khan, R.A.*, *Anwaar, M.U.**, *Kaddah, O.**, *Han, Z**. *and Kleinsteuber, M., 2021, September. Unsupervised Learning of Joint Embeddings for Node Representation and Community Detection. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 19-35). Springer, Cham.*

In this chapter we simultaneously look at two important tasks in graph analysis:

- **Network Embedding**, which deals with learning the node embeddings of a network. The quality of network embedding is measured by how well it performs for the downstream tasks such as graph visualization [79, 110–112], node clustering, and transductive node classification [81, 83].
- **Community detection**, where the objective is to cluster nodes into multiple groups (communities). Each community is a set of densely connected nodes, according to some defined similarity function. The communities can be overlapping or non-overlapping, depending on whether they share some nodes or not. Several algorithmic [113,114] and probabilistic approaches [79,115–117] to community detection have been proposed. The task of community detection is evaluated by employing some objective function that quantifies the inta-community similarities against the inter-community similarities.

In the literature, these tasks are usually treated separately. Although the standard graph embedding methods capture the basic connectivity, the learning of the node embeddings is independent of community detection. For instance, a simple approach can be to get the node embeddings via DeepWalk [63] and get community assignments for each node by using k-means or the Gaussian mixture model. Looking from the other perspective, methods like Bigclam [118], that focus on finding the community structure in the dataset, perform poorly for node-representation tasks e.g. node classification. This motivates us to study the approaches that jointly learn community-aware node embeddings.

Recently several approaches, like CNRL [58], ComE [59], vGraph [60] etc, have been proposed to learn the node embeddings and detect communities simultaneously in a unified framework. Several studies have shown that community detection is

improved by incorporating the node representation in the learning process [81, 119]. The intuition is that the global structure of graphs learned during community detection can provide useful context for node embeddings and vice versa.

The joint learning methods (CNRL, ComE and vGraph) learn two embeddings for each node. One node embedding is used for the node representation task. The second node embedding is the "context" embedding of the node which aids in community detection. As CNRL and ComE are based on Skip-Gram [120] and DeepWalk [63], they inherit "context" embedding from it for learning the neighborhood information of the node. vGraph also requires two node embeddings for parameterizing two different distributions. In contrast, we propose learning a single community-aware node representation that is directly used for both tasks.

In this work, we propose an efficient generative model called **J-ENC** for jointly learning both community detection and node representation. The underlying intuition behind J-ENC is that every node can be a member of one or more communities. However, the node embeddings should be learned in such a way that connected nodes are "closer" to each other than unconnected nodes. Moreover, connected nodes should have similar community assignments. Formally, we assume that for i-th node, the node embeddings $\mathbf{z}_i$ are generated from a prior distribution $p(\mathbf{z})$. Given $\mathbf{z}_i$, the community assignments c_i are sampled from $p(c_i|\mathbf{z}_i)$, which is parameterized by node and community embeddings. In order to generate an edge (i, j), we sample another node embedding $\mathbf{z}_j$ from $p(\mathbf{z})$ and respective community assignment c_j from $p(c_j|\mathbf{z}_j)$. Afterward, the node embeddings and the respective community assignments of node pairs are fed to a decoder. The decoder ensures that embeddings of both the nodes and the communities of connected nodes share high similarity. This enables learning such node embeddings that are useful for both community detection and node representation tasks.

We validate the effectiveness of our approach on several real-world graph datasets. In section 4.3, we show empirically that J-ENC is able to outperform the baseline methods including the direct competitors on all three tasks i.e. node classification, overlapping community detection and non-overlapping community detection. Furthermore, we compare the computational cost of training different algorithms. J-ENC is up to 40x more time-efficient than its competitors. We also conduct hyperparameter sensitivity analysis which demonstrates the robustness of our approach. Our main contributions are summarized below:

- We propose an efficient generative model called **J-ENC** for joint community detection and node representation learning.
- We adopt a novel approach and argue that a single node embedding is sufficient for learning both the representation of the node itself and its context.
- Training J-ENC is extremely time-efficient in comparison to its competitors.

We restrict ourselves to the case of homogeneous graph datasets i.e. the networks where the nodes are of the same type. This setting also ensures that the edges

are also of the same type. A common example of such a graph is a social network, where a node represents a user and an edge indicates a connection between two users.

4.1 Related Work

4.1.1 Community Detection

Early community detection algorithms are inspired by clustering algorithms [121]. For instance, spectral clustering [73] is applied to the graph Laplacian matrix for extracting the communities. Similarly, several matrix factorization-based methods have been proposed to tackle the community detection problem. For example, Bigclam [118] treats the problem as a non-negative matrix factorization (NMF) task. Another method CESNA [117] extends Bigclam by modeling the interaction between the network structure and the node attributes. Some generative models, like vGraph [60], Circles [116] etc, have also been proposed to detect communities in a graph.

4.1.2 Node Representation Learning

Many successful algorithms which learn node representation in an unsupervised way are based on random walk objectives [48, 63, 75]. Some known issues with random-walk-based methods (e.g. DeepWalk, node2vec etc) are: (1) They sacrifice the structural information of the graph by putting over-emphasis on the proximity information [84] and (2) great dependence of the performance on hyperparameters (walk-length, number of hops etc) [63, 75]. Some interesting GCN-based approaches include graph autoencoders e.g. GAE and VGAE [39] and DGI [49].

4.1.3 Joint Community Detection and Node Representation Learning

In the literature, several attempts have been made to tackle both of these tasks in a single framework. Most of these methods propose an alternate optimization process, i.e. learn node embeddings and improve community assignments with them and vice versa [58, 59]. Some approaches (CNRL [58], ComE [59]) are inspired by random walk, thus they inherit the issues discussed above. Others, like GEMSEC [122], are limited to the detection of non-overlapping communities. Some generative models like CommunityGAN [123] and vGraph [60] also jointly learn community assignments and node embeddings. CNRL, ComE and vGraph require learning two embeddings for each node for simultaneously tackling the two tasks. Unlike them, J-ENC learns a single community-aware node representation which is directly used for both tasks.

It is pertinent to highlight that although both vGraph and J-ENC adopt a variational approach but the underlying models are quite different. vGraph assumes

that each node can be represented as a mixture of multiple communities and is described by a multinomial distribution over communities, whereas J-ENC models the node embedding by a single distribution. For a given node, vGraph, first draws a community assignment and then a connected neighbor node is generated based on the assignment. Whereas, J-ENC draws the node embedding from the prior distribution and then community assignment is conditioned on a single node only. In simple terms, vGraph also needs edge information in the generative process whereas J-ENC does not require it. J-ENC relies on the decoder to ensure that embeddings of the connected nodes and their communities share high similarity with each other.

4.2 Methodology

4.2.1 Problem Formulation

Suppose an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ and a matrix $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ of F-dimensional node features, N being the number of nodes. Given K as the number of communities, we aim to jointly learn the node embeddings and the community embeddings following a variational approach such that:

- One or more communities can be assigned to every node.
- The node embeddings can be used for both community detection and node classification.

4.2.2 Variational Model

Let the random variables $\mathbf{z}_i \in \mathbb{R}^d$ and c_i respectively denote the latent node embeddings and community assignment for i-th node. The probability of the constructing $\mathbf{A}$ is modelled as:

$$p(\boldsymbol{A}) = \int \sum_{\boldsymbol{c}} p_\theta(\boldsymbol{Z}, \boldsymbol{c}, \boldsymbol{A}) d\boldsymbol{Z}, \tag{4.1}$$

where θ denotes the model parameters, $\boldsymbol{c}$ stacks the community assignments as $\boldsymbol{c} = [c_1, c_2, \cdots, c_N]^T$ and the matrix $\boldsymbol{Z}$ stacks the node embeddings $\mathbf{z}_i$. The joint distribution in equation 4.1 is factorized as

$$p(\boldsymbol{Z}, \boldsymbol{c}, \boldsymbol{A}) = p(\boldsymbol{Z})\ p_\theta(\boldsymbol{c}|\boldsymbol{Z})\ p_\theta(\boldsymbol{A}|\boldsymbol{c}, \boldsymbol{Z}). \tag{4.2}$$

Let us denote elements of $\boldsymbol{A}$ by a_{ij}. Following existing approaches [39, 93], we consider $\mathbf{z}_i$ to be *i.i.d.* random variables. The conditional random variables $c_i|\mathbf{z}_i$ are also assumed *i.i.d.* The reconstruction of a_{ij} depends upon the node embeddings $\mathbf{z}_i$

and $\mathbf{z}_j$, as well as the community assignments c_i and c_j. The underlying intuition comes from the observation that the connected nodes have a high probability of falling into the same community. Since $\mathbf{z}_i$ have been assumed *i.i.d.*, we need to reconstruct a_{ij} in a way to ensure the dependence between the connected nodes and the communities chosen by them. Following these assumptions, the joint distributions in equation 4.2 can be factorized as

$$p(\boldsymbol{Z}) = \prod_{i=1}^{N} p(\mathbf{z}_i) \tag{4.3}$$

$$p_\theta(\boldsymbol{c}|\boldsymbol{Z}) = \prod_{i=1}^{N} p_\theta(c_i|\mathbf{z}_i) \tag{4.4}$$

$$p_\theta(\boldsymbol{A}|\boldsymbol{c}, \boldsymbol{Z}) = \prod_{i,j} p_\theta(\mathrm{a}_{ij}|c_i, c_j, \mathbf{z}_i, \mathbf{z}_j), \tag{4.5}$$

where equation 4.5 assumes that the *edge decoder* $p_\theta(\mathrm{a}_{ij}|c_i, c_j, \mathbf{z}_i, \mathbf{z}_j)$ depends only on $c_i, c_j, \mathbf{z}_i$ and $\mathbf{z}_j$.

We aim to learn the model parameters θ such that $\log(p_\theta(\boldsymbol{A}))$ is maximized. In order to ensure computational tractability, we introduce the approximate posterior

$$q_\phi(\boldsymbol{Z}, \boldsymbol{c}|\mathcal{I}) = \prod_i q_\phi(\mathbf{z}_i, c_i|\mathcal{I}) \tag{4.6}$$

$$= \prod_i q_\phi(\mathbf{z}_i|\mathcal{I}) q_\phi(c_i|\mathbf{z}_i, \mathcal{I}), \tag{4.7}$$

where ϕ denotes the parameters of the approximate posterior. We set $\mathcal{I} = (\boldsymbol{A}, \boldsymbol{X})$ if node features are available, otherwise $\mathcal{I} = \boldsymbol{A}$. The objective now takes the form

$$\log(p_\theta(\mathbf{A})) = \log\left(\mathbb{E}_{q_\phi(\mathbf{Z},\mathbf{c}|\mathcal{I})}\left\{\frac{p(\mathbf{Z})\ p_\theta(\mathbf{c}|\mathbf{Z})\ p_\theta(\mathbf{A}|\mathbf{c},\mathbf{Z})}{q_\phi(\mathbf{Z}|\mathcal{I}) q_\phi(\mathbf{c}|\mathbf{Z},\mathcal{I})}\right\}\right) \tag{4.8}$$

$$\geq \mathbb{E}_{q_\phi(\mathbf{Z},\mathbf{c}|\mathcal{I})}\left\{\log\left(\frac{p(\mathbf{Z})\ p_\theta(\mathbf{c}|\mathbf{Z})\ p_\theta(\mathbf{A}|\mathbf{c},\mathbf{Z})}{q_\phi(\mathbf{Z}|\mathcal{I}) q_\phi(\mathbf{c}|\mathbf{Z},\mathcal{I})}\right)\right\}, \tag{4.9}$$

where 4.9 follows from Jensen's Inequality. So we maximize, with respect to the parameters θ and ϕ, the corresponding ELBO bound $\mathcal{L}_{\text{ELBO}}$ in 4.9, given by

$$\begin{aligned}\mathcal{L}_{\text{ELBO}} \approx & \underbrace{-\sum_{i=1}^{N} D_{KL}\Big(q_\phi(\mathbf{z}_i|\mathcal{I}) \;||\; p(\mathbf{z}_i)\Big)}_{L_{\text{enc}}} \\ & \underbrace{-\sum_{i=1}^{N}\frac{1}{\mathrm{J}}\sum_{j=1}^{\mathrm{J}} D_{KL}\Big(q_\phi(c_i|\mathbf{z}_i^{(j)},\mathcal{I}) \;||\; p_\theta(c_i|\mathbf{z}_i^{(j)})\Big)}_{L_c} \\ & \underbrace{+\sum_{(i,j)\in\mathcal{E}} \mathbb{E}_{(\mathbf{z}_i,\mathbf{z}_j,c_i,c_j)\sim q_\phi(\mathbf{z}_i,\mathbf{z}_j,c_i,c_j|\mathcal{I})}\Big\{\log\Big(p_\theta(\mathrm{a}_{ij}|c_i,c_j,\mathbf{z}_i,\mathbf{z}_j)\Big)\Big\}}_{-L_{\text{recon}}}, \end{aligned} \tag{4.10}$$

where $D_{KL}(\cdot||\cdot)$ represents the KL-divergence between two distributions. The detailed derivation of equation 4.10 is provided in appendix B. Equation 4.10 contains three major summation terms as indicated by the braces under these terms.

1. L_{enc} refers to the encoder loss. Minimizing this term minimizes the mismatch between the approximate posterior and the prior distributions of the node embeddings.

2. L_{c} gives the mismatch between the categorical distributions governing the community assignments. Minimizing this term ensures that the community assignments c_i take into consideration the respective node embeddings.

3. The third term is the negative of the reconstruction loss or L_{recon}. It is the negative of binary cross-entropy (BCE) between the input and the reconstructed edges.

Instead of maximizing the ELBO bound, we can minimize the corresponding loss, which we refer to as the variational loss or L_{var}, given by

$$L = -\mathcal{L}_{\text{ELBO}} = L_{\text{enc}} + L_{\text{c}} + L_{\text{recon}} \tag{4.11}$$

4.2.3 Choice of Distributions

Distributions Involved in L_{enc}: In equation 4.3, $p(\mathbf{z}_i)$ is chosen to be the standard gaussian distribution for all i. The corresponding approximate posterior $q_\phi(\mathbf{z}_i|\mathcal{I})$ in equation 4.7, is also chosen to be a gaussian, with the parameters (mean $\boldsymbol{\mu}_i$ and variance $\boldsymbol{\sigma}_i^2$) learned by the encoder block. i.e.

$$q_\phi(\mathbf{z}_i|\mathcal{I}) = \mathcal{N}\big(\boldsymbol{\mu}_i(\mathcal{I}), \mathrm{diag}(\boldsymbol{\sigma}^2{}_i(\mathcal{I}))\big). \tag{4.12}$$

Distributions Involved in L_c: For parameterizing $p_\theta(c_i|\mathbf{z}_i)$ in equation 4.4, we introduce community embeddings $\{\mathbf{g}_1, \cdots, \mathbf{g}_K\}$; $\mathbf{g}_k \in \mathbb{R}^d$. The distribution $p_\theta(c_i|\mathbf{z}_i)$ is then modelled as the softmax of dot products of $\mathbf{z}_i$ with $\mathbf{g}_k$, i.e.

$$p_\theta(c_i = k|\mathbf{z}_i) = \text{softmax}(< \mathbf{z}_i, \mathbf{g}_k >). \tag{4.13}$$

The softmax is over K community embeddings to ensure that $p_\theta(c_i|\mathbf{z}_i)$ is a distribution. The corresponding approximate posterior $q_\phi(c_i = k|\mathbf{z}_i, \mathcal{I})$ in equation 4.7 is affected by the node embedding $\mathbf{z}_i$ as well as the neighborhood. To design this, our intuition is to consider the similarity of $\mathbf{g}_k$ with the embedding $\mathbf{z}_i$ as well as with the embeddings of the neighbors of the i-th node. The overall similarity with neighbors is mathematically formulated as the average of the dot products of their embeddings. Afterward, a hyperparameter α is introduced to control the bias between the effect of $\mathbf{z}_i$ and the set $\mathcal{N}_i$ of the neighbors of the i-th node. Finally, a softmax is applied, i.e.

$$q_\phi(c_i = k|\mathbf{z}_i, \mathcal{I}) = \text{softmax}\Big(\alpha < \mathbf{z}_i, \mathbf{g}_k > +(1-\alpha)\frac{1}{|\mathcal{N}_i|}\sum_{j \in \mathcal{N}_i} < \mathbf{z}_j, \mathbf{g}_k > \Big). \tag{4.14}$$

Hence, equation 4.14 ensures that graph structure information is employed to learn community assignments instead of relying on an extraneous node embedding as done in [59, 60].

Distributions Involved in L_{recon}: The distribution $q_\phi(\mathbf{z}_i, \mathbf{z}_j, c_i, c_j|\mathcal{I})$ in the third term of equation 4.10 is factorized into two conditionally independent distributions i.e.

$$q_\phi(\mathbf{z}_i, \mathbf{z}_j, c_i, c_j|\mathcal{I}) = q_\phi(\mathbf{z}_i, c_i|\mathcal{I})q_\phi(\mathbf{z}_j, c_j|\mathcal{I}). \tag{4.15}$$

where $q_\phi(\mathbf{z}_i, c_i|\mathcal{I})$ factorization and the related distributions have been given in equation 4.7, 4.12 and 4.14.

Finally, the *edge-decoder* in equation 4.5 is modeled to maximize the probability of connected nodes having same community assignments, i.e.,

$$p_\theta(\mathrm{a}_{ij}|c_i = \ell, c_j = m, \mathbf{z}_i, \mathbf{z}_j) = \frac{\text{sigmoid}(< \mathbf{z}_i, \mathbf{g}_m >) + \text{sigmoid}(< \mathbf{z}_j, \mathbf{g}_\ell >)}{2}. \tag{4.16}$$

The primary objective of the variational module in equation 4.11 is not to directly optimize the community assignments, but to preserve edge information in a community-aware fashion. So the information in the community embeddings, the node embeddings, and the community assignments is simultaneously incorporated by equation 4.16. This formation forces the community embeddings of the connected nodes to be similar and vice versa. On one hand, this helps in learning better node representations by leveraging the global information about the graph structure

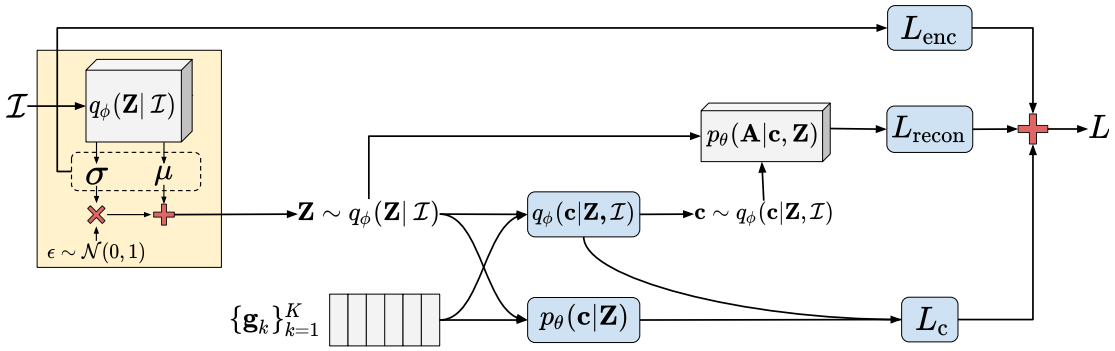


Figure 4.1: Block diagram of J-ENC . The encoder (in yellow) maps $\mathcal{I}$ to $\mathbf{Z}$, which, along with the community embeddings $\mathbf{g}_k$, is used to sample $\mathbf{c}$. Both $\mathbf{Z}$ and $\mathbf{c}$ are then fed to the edge-decoder to reconstruct $\mathbf{A}$. The trainable blocks of the architecture are colored gray, and the deterministic functions are colored blue.

via community assignments. On the other hand, this also refines the community embeddings by exploiting the local graph structure via node embeddings and edge information.

4.2.4 Practical Aspects

Following the pattern in section 4.2.3, we discuss the practical considerations related to the different loss terms in equation 4.10.

Conbsiderations Related to L_{enc}: For computational stability, we learn $\log(\sigma)$ instead of variance in 4.7. The parameters of $q_\phi(\mathbf{z}_i|\mathcal{I})$ can be learned by any encoder network e.g. graph convolutional network [40], graph attention network [46], GraphSAGE [48] or even two matrices to learn $\boldsymbol{\mu}_i(\mathcal{I})$ and $\text{diag}(\boldsymbol{\sigma}^2{}_i(\mathcal{I}))$. Samples are then generated using reparameterization trick [19].

Conbsiderations Related to L_c: Since community assignment follows a categorical distribution, we use Gumbel-softmax [124] for efficient backpropagation of the gradients.

Conbsiderations Related to L_{recon}: The third term in equation 4.10 is estimated in practice using the samples generated by the approximate posterior. This term is equivalent to the negative binary cross-entropy (BCE) loss between observed edges and reconstructed edges. The decoder block requires both positive and negative edges for learning the respective distribution in equation 4.16. Hence we follow the current approaches, e.g., [39, 93] and sample an equal number of negative edges from $\mathbf{A}$ to provide at the decoder input along with the positive edges.

4.2.5 Inference

For inference, non-overlapping community assignment can be obtained for i-th node as

$$\mathcal{C}_i = \underset{k \in \{1, \cdots, K\}}{\arg\max} \; q_\phi(c_i = k | \mathbf{z}_i, \mathcal{I}). \tag{4.17}$$

To get overlapping community assignments for i-th node, we can threshold its weighted probability vector at ϵ, a hyperparameter, as follows

$$\mathcal{C}_i = \Big\{ k \;\Big|\; \frac{q_\phi(c_i = k | \mathbf{z}_i, \mathcal{I})}{\max_\ell q_\phi(c_i = \ell | \mathbf{z}_i, \mathcal{I})} \geq \epsilon \Big\}, \quad \epsilon \in [0, 1]. \tag{4.18}$$

4.2.6 Complexity

The computation of dot products for all combinations of node and community embeddings takes $\mathcal{O}(NKd)$ time. Solving equation 4.14 further requires the calculation of the mean of dot products over the neighborhood for every node, which takes $\mathcal{O}(|\mathcal{E}|K)$ computations overall as we traverse every edge for every community. Finally, we need softmax over all communities for every node in equation 4.13 and equation 4.14 which takes $\mathcal{O}(NK)$ time. equation 4.16 takes $\mathcal{O}(|\mathcal{E}|)$ time for all edges as we have already calculated the dot products. As a result, the overall complexity becomes $\mathcal{O}(|\mathcal{E}|K + NKd)$. This complexity is quite low compared to other algorithms designed to achieve similar goals [59, 125].

4.3 Experiments

4.3.1 Synthetic Example

We start with a synthetic dataset, consisting of 3 communities with 5 points per community. This dataset is actually a random partition graph generated by the python package `networkx`. The encoder simply consists of two matrices that give $\boldsymbol{\mu}_i(\mathcal{I})$ and $\text{diag}(\boldsymbol{\sigma}^2{}_i(\mathcal{I}))$. The results of the community assignments discovered by J-ENC are given in figure 4.2, where the node sizes are reciprocal to the confidence of J-ENC in the community assignments. We choose 3 communities for demonstration because the probabilistic community assignments in such case can be thought of as `rgb` values for coloring the nodes. It can be seen that J-ENC discovers the correct community structure. However, the two bigger nodes in the center can be assigned to more than one community as J-ENC is not very confident in the case of these nodes. This is evident from the colors which are a mix of red, green and blue. We now proceed to the experiments on real-world datasets.

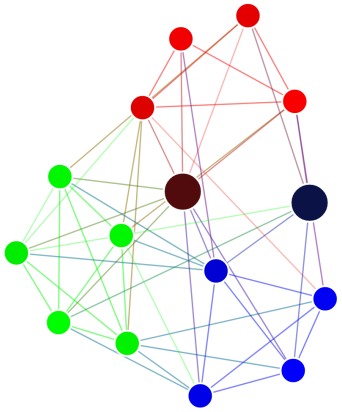


Figure 4.2: Visualization of community assignments discovered by J-ENC in the synthetic dataset of 15 points divided into three communities.

4.3.2 Datasets

We have selected 18 different datasets ranging from 270 to 126,842 edges. For non-overlapping community detection and node classification, we use 5 the citation datasets [108, 126]. The remaining datasets [116, 127], used for overlapping community detection, are taken from SNAP repository [128]. Following [60], we take 5 biggest ground truth communities for youtube, amazon and dblp. Moreover, we also analyze the case of a large number of communities. For this purpose, we prepare two subsets of the amazon dataset by randomly selecting 500 and 1000 communities from the 2000 smallest communities in the amazon dataset.

4.3.3 Baselines

For overlapping community detection, we compare with the following competitive baselines: **MNMF** [79] learns community membership distribution by using joint non-negative matrix factorization with modularity based regularization. **BIGCLAM** [118] also formulates community detection as a non-negative matrix factorization (NMF) task. It simultaneously optimizes the model likelihood of observed links and learns the latent factors which represent community affiliations of nodes. **CESNA** [117] extends BIGCLAM by statistically modeling the interaction between the network structure and the node attributes. **Circles** [116] introduces a generative model for community detection in ego networks by learning node similarity metrics for every community. **SVI** [115] formulates membership of nodes in multiple

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	K	$\lvert F\rvert$	Overlap
CiteSeer	3327	9104	6	3703	N
CiteSeer-full	4230	10674	6	602	N
Cora	2708	10556	7	1433	N
Cora-ML	2995	16316	7	2879	N
Cora-full	19793	126842	70	8710	N
fb0	333	2519	24	N/A	Y
fb107	1034	26749	9	N/A	Y
fb1684	786	14024	17	N/A	Y
fb1912	747	30025	46	N/A	Y
fb3437	534	4813	32	N/A	Y
fb348	224	3192	14	N/A	Y
fb414	150	1693	7	N/A	Y
fb698	61	270	13	N/A	Y
youtube	5346	24121	5	N/A	Y
amazon	794	2109	5	N/A	Y
amazon500	1113	3496	500	N/A	Y
amazon1000	1540	4488	1000	N/A	Y
dblp	24493	89063	5	N/A	Y

Table 4.1: Every dataset has $\lvert\mathcal{V}\rvert$ nodes, $\lvert\mathcal{E}\rvert$ edges, K communities and $\lvert F\rvert$ features. $\lvert F\rvert$ = N/A means that either the features were missing or not used.

communities by a Bayesian model of networks. **vGraph** [60] simultaneously learns node embeddings and community assignments by modeling the nodes as being generated from a mixture of communities. **vGraph+**, a variant further incorporates regularization to weigh local connectivity. **ComE** [59] jointly learns community and node embeddings by using gaussian mixture model formulation. **CNRL** [58] enhances the random walk sequences (generated by DeepWalk, node2vec etc) to jointly learn community and node embeddings. **CommunityGAN** (ComGAN)is a generative adversarial model for learning node embeddings such that the entries of the embedding vector of each node refer to the membership strength of the node to different communities. Lastly, we compare the results with the communities obtained by applying k-means to the learned embeddings of **DGI** [49].

For non-overlapping community detection and node classification, in addition to MNMF, DGI, CNRL, CommunityGAN, vGraph and ComE, we compare J-ENC with the following baselines: **DeepWalk** [63] makes use of SkipGram [120] and truncated random walks on the network to learn node embeddings. **LINE** [83] learns node embeddings while attempting to preserve first and second-order proximities of nodes. **Node2Vec** [75] learns the embeddings using biased random walk while aiming to preserve network neighborhoods of nodes. **Graph Autoencoder (GAE)** [39] extends the idea of autoencoders to graph datasets. We also include its variational counterpart i.e. **VGAE**. **GEMSEC** is a sequence

sampling-based learning model which aims to jointly learn the node embeddings and clustering assignments.

4.3.4 Settings

For overlapping community detection, we learn mean and log-variance matrices of 16-dimensional node embeddings. We set $\alpha = 0.9$ and $\epsilon = 0.3$ in all our experiments. Following [39], we first pre-train a variational graph autoencoder. We perform gradient descent with Adam optimizer [76] and learning rate = 0.01. Community assignments are obtained using equation 4.18. For the baselines, we employ the results reported by [60]. For evaluating the performance, we use *F1-score* and *Jaccard similarity*.

For non-overlapping community detection, since the default implementations of most the baselines use 128 dimensional embeddings, for we use $d = 128$ for fair comparison. equation 4.17 is used for community assignments. For vGraph, we use the code provided by the authors. We employ *normalized mutual information (NMI)* and *adjusted random index (ARI)* as evaluation metrics.

For node classification, we follow the training split used in various previous works [40, 49, 108], i.e. 20 nodes per class for training. We train logistic regression using LIBLINEAR [129] solver as our classifier and report the evaluation results on the rest of the nodes. For the algorithms that do not use node features, we train the classifier by appending the raw node features with the learned embeddings. For evaluation, we use *F1-macro* and *F1-micro* scores.

All the reported results are the average over five runs. Further implementation details can be found in [130].

4.3.5 Discussion of Results

Tables 4.2 and 4.3 summarize the results of the performance comparison for the overlapping community detection task.

First, we note that our proposed method J-ENC outperforms the competitors on all datasets in terms of Jaccard score as well as F1-score, with the dataset (*fb0*) being the only exception where J-ENC is the second best. These results demonstrate the capability of J-ENC to learn multiple community assignments quite well and hence reinforce our intuition behind the design of equation 4.14.

Second, we observe that there is no consistent performing algorithm among the competitive methods. That is, excluding J-ENC , the best performance is achieved by vGraph/vGraph+ on 5, ComGAN on 4 and ComE on 3 out of 13 datasets in terms of F1-score. A similar trend can be seen in Jaccard Similarity. It is worth noting that all the methods, which achieve the second-best performance, are solving the task of community detection and node representation learning jointly.

Dataset	MNMF	Bigclam	CESNA	Circles	SVI	vGraph	vGraph+	ComE	CNRL	ComGan	DGI	J-ENC
fb0	14.4	29.5	28.1	28.6	28.1	24.4	26.1	31.1	11.5	**35.0**	27.4	34.7
fb107	12.6	39.3	37.3	24.7	26.9	28.2	31.8	39.7	20.2	47.5	35.8	**59.7**
fb1684	12.2	50.4	51.2	28.9	35.9	42.3	43.8	52.9	38.5	47.6	42.8	**56.4**
fb1912	14.9	34.9	34.7	26.2	28.0	25.8	37.5	28.7	8.0	35.6	32.6	**45.8**
fb3437	13.7	19.9	20.1	10.1	15.4	20.9	22.7	21.3	3.9	39.3	19.7	**50.2**
fb348	20.0	49.6	53.8	51.8	46.1	55.4	53.1	46.2	34.1	55.8	54.7	**58.2**
fb414	22.1	58.9	60.1	48.4	38.9	64.7	66.9	55.3	25.3	43.9	56.9	**69.6**
fb698	26.6	54.2	58.7	35.2	40.3	54.0	59.5	45.8	16.4	58.2	52.2	**64.0**
Youtube	59.9	43.7	38.4	36.0	41.4	50.7	52.2	65.5	51.4	43.6	47.8	**67.3**
Amazon	38.2	46.4	46.8	53.3	47.3	53.3	53.2	50.1	53.5	51.4	44.7	**58.1**
Amazon500	30.1	52.2	57.3	46.2	41.9	61.2	60.4	59.8	38.4	59.3	33.8	**67.6**
Amazon1000	19.3	28.6	30.8	25.9	21.6	54.3	47.3	50.3	27.1	52.7	37.7	**60.5**
Dblp	21.8	23.6	35.9	36.2	33.7	39.3	39.9	47.1	46.8	34.9	44.0	**53.9**

Table 4.2: F1 scores (%) for overlapping communities. Best and second best values are bold and blue respectively.

Dataset	MNMF	Bigclam	CESNA	Circles	SVI	vGraph	vGraph+	ComE	CNRL	ComGan	DGI	J-ENC
fb0	08.0	18.5	17.3	18.6	17.6	14.6	15.9	19.5	06.8	24.1	16.8	**24.7**
fb107	06.9	27.5	27.0	15.5	17.2	18.3	21.7	28.7	11.9	38.5	25.3	**46.8**
fb1684	06.6	38.0	38.7	18.7	24.7	29.2	32.7	40.3	25.8	37.9	38.8	**42.5**
fb1912	08.4	24.1	23.9	16.7	20.1	18.6	28.0	18.5	04.6	13.5	22.5	**37.3**
fb3437	07.7	11.5	11.7	05.5	09.0	12.0	13.3	12.5	02.0	33.4	11.6	**36.2**
fb348	11.3	35.9	40.0	39.3	33.6	41.0	40.5	34.4	21.7	23.2	41.8	**43.5**
fb414	12.8	47.1	47.3	34.2	29.3	51.8	55.9	42.2	15.4	53.6	46.4	**58.4**
fb698	16.0	41.9	45.9	22.6	30.0	43.6	47.7	33.8	09.6	46.9	42.1	**50.4**
Youtube	46.7	29.3	24.2	22.1	28.7	34.3	34.8	52.5	35.5	44.0	32.7	**53.3**
Amazon	25.2	35.1	35.0	36.7	36.4	36.9	36.9	34.6	38.7	38.0	29.1	**41.9**
Amazon500	20.8	51.2	53.8	47.2	45.0	59.1	59.6	58.4	41.1	57.3	23.3	**64.9**
Amazon1000	20.3	26.8	28.9	24.9	23.6	54.3	49.7	52.0	26.9	54.1	23.2	**57.1**
Dblp	20.9	13.8	22.3	23.3	20.9	25.0	25.1	27.9	32.8	25.0	29.2	**37.3**

Table 4.3: Jaccard scores (%) for overlapping communities. Best and second best values are bold and blue respectively.

	Dataset	MNMF	DeepWalk	LINE	Node2Vec	GAE	VGAE	DGI	GEMSEC	CNRL	ComGAN	vGraph	ComE	J-ENC
NMI(%)	**CiteSeer**	14.1	08.8	08.7	14.9	17.4	16.3	37.8	11.8	13.6	03.2	09.0	18.8	**38.5**
	CiteSeer-full	09.4	15.4	13.0	22.3	55.1	48.4	56.7	11.1	23.3	16.2	07.6	32.8	**59.0**
	Cora	19.7	39.7	32.8	39.7	39.7	40.8	50.1	27.4	39.4	05.7	26.4	39.6	**52.7**
	Cora-ML	37.8	43.2	42.3	39.6	48.3	48.3	46.2	18.1	42.9	11.5	29.8	47.6	**56.3**
	Cora-full	42.0	48.5	40.3	48.1	48.3	47.0	39.9	10.0	47.7	15.0	41.7	51.2	**55.2**
ARI(%)	**CiteSeer**	02.6	09.5	03.3	08.1	14.1	10.1	**38.1**	00.6	12.8	01.2	05.1	13.8	35.2
	CiteSeer-full	00.4	16.4	03.7	10.5	50.6	40.6	50.8	01.0	20.2	04.9	04.2	20.9	**60.3**
	Cora	02.9	31.2	14.9	25.8	29.3	34.7	44.7	04.8	31.9	03.2	12.7	34.2	**45.1**
	Cora-ML	24.1	33.9	32.7	27.9	41.8	42.5	42.1	01.0	32.5	06.7	21.6	37.2	**49.8**
	Cora-full	06.1	22.5	11.7	18.8	18.3	17.9	12.1	00.2	22.9	00.6	14.9	19.7	**28.8**

Table 4.4: Non-overlapping community detection results. Best and second best values are bold and blue respectively.

	Dataset	MNMF	DeepWalk	LINE	Node2Vec	GAE	VGAE	DGI	GEMSEC	CNRL	ComGAN	vGraph	ComE	J-ENC
F1-macro(%)	**CiteSeer**	57.4	49.0	55.0	55.2	57.9	59.1	62.6	37.5	50.0	55.9	30.8	59.6	**64.8**
	CiteSeer-full	68.6	56.6	60.2	61.0	79.9	74.4	**82.1**	53.3	58.0	65.7	28.5	69.9	76.8
	Cora	60.9	69.7	68.0	71.3	71.2	70.4	71.1	60.3	70.4	56.6	44.7	71.6	**73.1**
	Cora-ML	64.2	75.8	75.3	78.4	76.5	75.2	72.6	70.6	77.8	62.5	59.8	78.5	**80.2**
	Cora-full	30.4	41.7	39.4	42.3	36.6	32.4	16.5	35.8	41.3	27.7	33.4	42.2	**43.1**
F1-micro(%)	**CiteSeer**	60.8	52.0	57.7	57.8	61.6	62.2	67.9	39.4	53.2	59.1	32.1	63.1	**68.2**
	CiteSeer-full	68.1	57.3	60.0	61.5	79.6	74.4	**81.8**	53.5	57.9	64.9	28.5	70.2	77.0
	Cora	62.7	70.2	68.3	71.4	73.5	72.0	73.3	59.4	70.4	58.5	44.6	74.2	**75.6**
	Cora-ML	64.2	75.6	74.6	78.6	77.6	76.4	75.4	72.5	78.4	62.8	62.3	79.5	**82.0**
	Cora-full	32.9	48.3	42.1	48.1	41.8	37.7	21.1	38.9	45.9	29.4	37.6	47.8	**49.6**

Table 4.5: Node classification results. Best and second best values are bold and blue respectively.

Third, we observe that vGraph+ results are generally better than vGraph. This is because vGraph+ incorporates a regularization term in the loss function which is based on Jaccard coefficients of connected nodes as edge weights. However, it should be noted that this prepossessing step is computationally expensive for densely connected graphs.

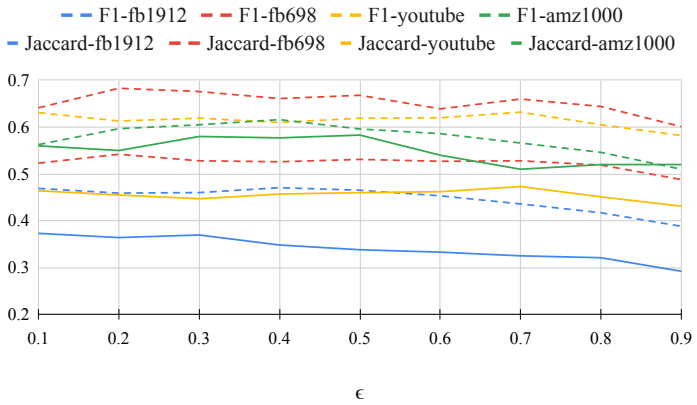
table 4.4 shows the results on non-overlapping community detection. First, we observe that MNMF, DeepWalk, LINE and Node2Vec provide a good baseline for the task. However, these methods are not able to achieve comparable performance on any dataset relative to the frameworks that treat the two tasks jointly. Second, J-ENC consistently outperforms all the competitors in NMI and ARI metrics, except for *CiteSeer* where it achieves second best ARI. Third, we observe that GCN-based models i.e. GAE, VGAE and DGI show competitive performance. That is, they achieve the second best performance in all the datasets except *CiteSeer*. In particular, DGI achieves second-best NMI results in 3 out of 5 datasets and 2 out of 5 datasets in terms of ARI. Nonetheless, DGI results are not very competitive in table 4.2 and table 4.3, showing that while DGI can be a good choice for learning node embeddings for attributed graphs with non-overlapping communities, it is not the best option for non-attributed graphs or overlapping communities.

The results for node classification are presented in table 4.5. J-ENC achieves best F1-micro and F1-macro scores on 4 out of 5 datasets. We also observe that GCN-based models i.e. GAE, VGAE and DGI show competitive performance, following the trend in results of table 4.4. Furthermore, we note that the node classification results of CommunityGan (ComGAN) are quite poor. We think a potential reason behind it is that the node embeddings are constrained to have the same dimensions as the number of communities. Hence, different components of the learned node embeddings simply represent the membership strengths of nodes for different communities. The linear classifiers may find it difficult to separate such vectors.

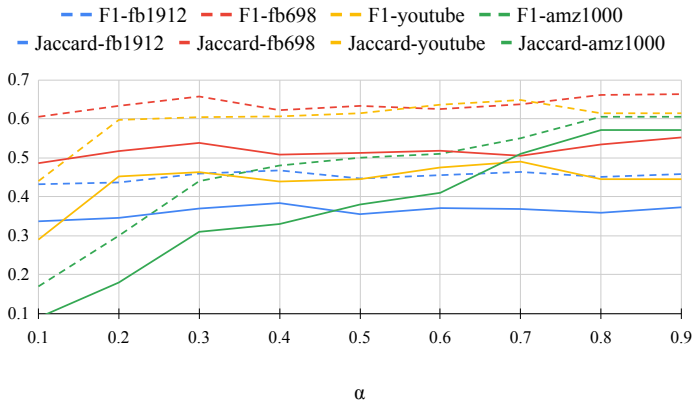
4.3.6 Hyperparameter Sensitivity

We study the dependence of J-ENC on ϵ and α by evaluating on four datasets of different sizes: *fb698*($N = 61$), *fb1912*($N = 747$), *amazon1000*(N=1540) and *youtube*($N = 5346$).

Effect of ϵ: We sweep for $\epsilon = \{0.1, 0.2, \cdots, 0.9\}$. For demonstrating effect of α, we fix $\epsilon = 0.3$ and sweep for $\alpha = \{0.1, 0.2, \cdots, 0.9\}$. The average results of five runs for ϵ and α are given in figure 4.3(a) and figure 4.3(b) respectively. Overall J-ENC is quite robust to the change in the values of ϵ and α. In the case of ϵ, we see a general trend of decrease in performance when the threshold ϵ is set quite high e.g. $\epsilon > 0.7$. This is because the datasets contain overlapping communities and a very high ϵ will cause the algorithm to give only the most probable community assignment instead of potentially providing multiple communities per node. However, for a large part of the sweep space, the results are almost consistent.



(a) Effect of ϵ. Overall a slight decrease in scores can be observed after $\epsilon = 0.7$ mark.



(b) Effect of α. The scores generally tend to decrease for small values of α.

Figure 4.3: Effect of hyperparameters on the performance. F1 and Jaccard scores are in solid and dashed lines respectively.

Effect of α: When ϵ is fixed and α is changed, the results are mostly consistent except when α is set to a low value. equation 4.14 shows that in such a case the node itself is almost neglected and J-ENC tends to assign communities based upon neighborhood only, which may cause a decrease in the performance. This effect is most visible in *amazon1000* dataset because it has only 1.54 points on average per community. This implies a decent chance for neighbors of a point of being in different communities. Thus, sole dependence on the neighbors will most likely result in poor results.

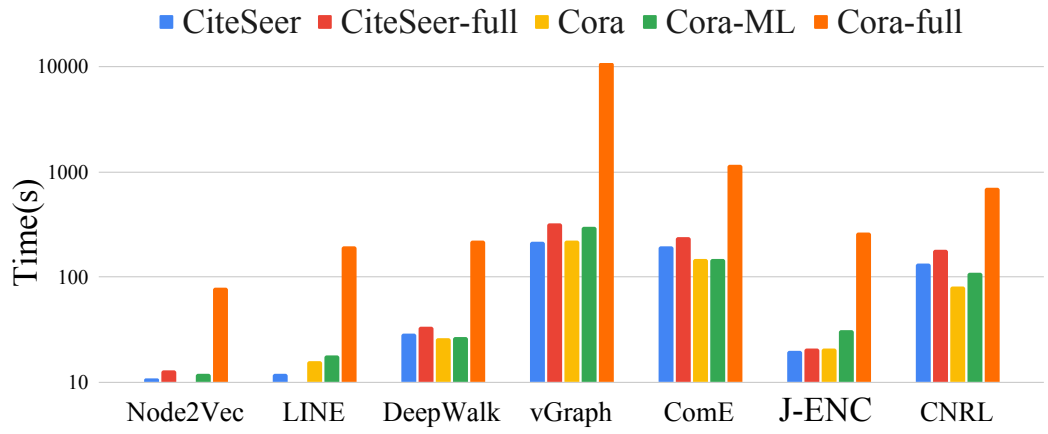


Figure 4.4: Comparison of running times of different algorithms. We can see that J-ENC outperforms the direct competitors. The time on the y-axis is in the log scale.

4.3.7 Training Time

Now we compare the training times of different algorithms in figure 4.4. As some of the baselines are more resource intensive than others, we select AWS instance type `g4dn.4xlarge` for a fair comparison of training times. For vGraph, we train for 1000 iterations and for J-ENC for 1500 iterations. For all other algorithms, we use the default parameters as used in section 4.3.4. We observe that the methods that simply output the node embeddings take relatively less time compared to the algorithms that jointly learn node representations and community assignments e.g J-ENC , vGraph and CNRL. Among these algorithms J-ENC is the most time efficient. It consistently trains in less time compared to its direct competitors. For instance, it is about 12 times faster than ComE for *CiteSeer-full* and about 40 times faster compared to vGraph for *Cora-full* dataset. This provides evidence for the lower computational complexity of J-ENC in Section 4.2.6.

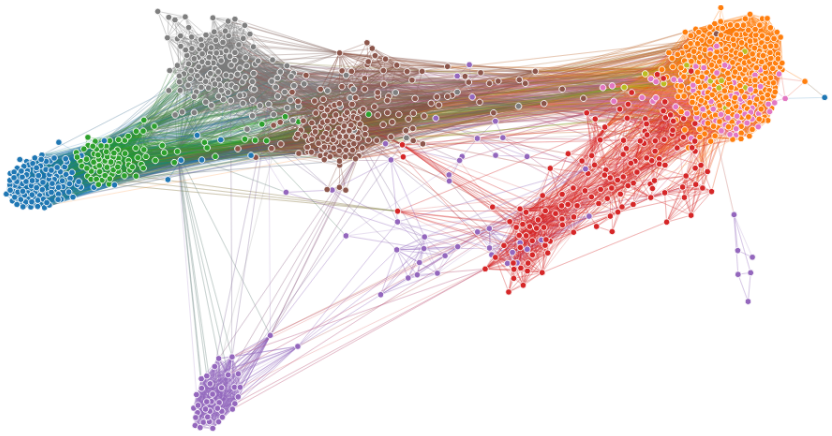
4.3.8 Visualization

Our experiments demonstrate that a single community-aware node embedding is sufficient to aid in both the node representation and community assignment tasks. This is also qualitatively demonstrated by graph visualizations of node embeddings (obtained via t-SNE [131]) and inferred communities for two datasets, fb107 and fb3437, presented in figure 4.5.

4.4 Conclusion

We propose a scalable generative method J-ENC to simultaneously perform community detection and node representation learning. Our novel approach learns a

single community-aware node embedding for both the representation of the node and its context. J-ENC is scalable due to its low complexity, i.e. $\mathcal{O}(|\mathcal{E}|K + NKd)$. The experiments on several graph datasets show that J-ENC consistently outperforms all the competitive baselines on node classification, overlapping community detection and non-overlapping community detection tasks. Moreover, training the J-ENC is highly time-efficient than its competitors.



(a) fb107

(b) fb3437

Figure 4.5: Graph visualization with community assignments (better viewed in color)

5 Case of Heterogeneous Networks

Based on the following peer-reviewed publication:

Khan, R.A.* *and Kleinsteuber, M., 2022, February. Cluster-Aware Heterogeneous Information Network Embedding. In Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (pp. 476-486).*

A large number of real-world phenomena express themselves in the form of heterogeneous information networks or HINs consisting of multiple node types and edges. For instance, figure 5.1 illustrates a sample HIN consisting of three types of nodes (authors, papers, and conferences) and two types of edges. A red edge indicates that an author *has written* a paper and blue edge models the relation that a paper *is published in* a conference. Compared to homogeneous networks (i.e., the networks consisting of only a single node type and edge type), HINs are able to convey a more comprehensive view of data by explicitly modeling the rich semantics and complex relations between multiple node types. The advantages of HINs over homogeneous networks have resulted in an increasing interest in the techniques related to HIN embedding [61]. The main idea of a network embedding is to project the graph nodes into a continuous low-dimensional vector space such that the structural properties of the network are preserved [25, 26]. Many HIN embedding methods follow *meta-paths* based approaches to preserve HIN structure [65,132,133], where a meta-path is a template indicating the sequence of relations between two node types in HIN. As an example, PAP and PCP in figure 5.1 are two meta-paths defining different semantic relations between two paper nodes. Although the HIN embedding techniques based on meta-paths have received considerable success, meta-path selection is still an open problem. Above that, the quality of HIN embedding highly depends on the selected meta-path [64]. To overcome this issue, either domain knowledge is utilized, which can be subjective and expensive, or some strategy is devised to fuse the information from predefined meta-paths [65–67].

As discussed in chapter 4, community detection is another important task for unsupervised network analysis, where the aim is to group similar graph nodes together. The most basic approach to achieve this can be to apply an off-the-shelf clustering algorithm, e.g., K-Means or Gaussian Mixture Model (GMM), on the learned network embedding. However, this usually results in suboptimal results because of treating the two highly correlated tasks, i.e., network embedding and

node clustering, in an independent way. There has been substantial evidence from the domain of euclidean data that the joint learning of latent embeddings and clustering assignments yields better results compared to when these two tasks are treated independently [54–57]. For homogeneous networks, there are some approaches, e.g., [58, 122, 123], that take motivation from the euclidean domain to jointly learn the network embedding and the node community assignments. However, no such approach exists for HINs to the best of our knowledge.

We propose an approach for **Va**riational **C**ommunity **A**ware **HIN E**mbedding, called VaCA-HINE to address the above challenges of meta-path selection and joint learning of HIN embedding and community assignments. Our approach makes use of two parts that simultaneously refine the target network embedding by optimizing their respective objectives. The first part employs a variational approach to preserve pairwise proximity in a community-aware manner by aiming that the connected nodes fall in the same community and vice versa. The second part utilizes a contrastive approach to preserve high-order HIN semantics by discriminating between real and corrupted instances of different meta-paths. VaCA-HINE is flexible in the sense that multiple meta-paths can be simultaneously used and they all aim to refine a single embedding. Hence, the HIN embedding is learned by jointly leveraging the information in pairwise relations, the community assignments, and the high-order HIN structure.

Our major contributions are summarized below:

- To the best of our knowledge, our work is the first to propose a unified approach for learning the HIN embedding and the node community assignments in a joint fashion.
- We propose a novel architecture that fuses together variational and contrastive approaches to preserve pairwise proximity as well as high-order HIN semantics by simultaneously employing multiple meta-paths, such that the meta-path selection problem is also mitigated.
- We show the efficacy of our approach by conducting community detection and downstream node classification experiments on multiple datasets.

5.1 Preliminaries

This section sets up the basic definitions and notations that will be followed in the subsequent sections.

Definition 5.1.1 (Heterogeneous Information Network)**.** A Heterogeneous Information Network or *HIN* is defined as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R}, \pi, \lambda\}$ with the sets of nodes and edges represented by $\mathcal{V}$ and $\mathcal{E}$ respectively. $\mathcal{A}$ and $\mathcal{R}$ denote the sets of node types and edge types respectively. In addition, a HIN has a node-type

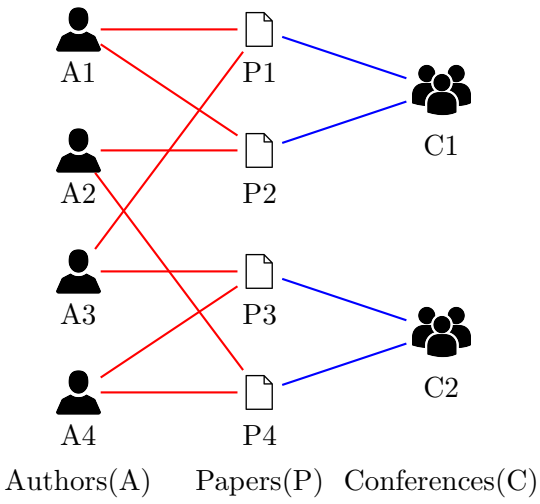


Figure 5.1: A sample HIN with three node types and two edge types.

mapping function $\pi : \mathcal{V} \to \mathcal{A}$ and an edge-type mapping function $\lambda : \mathcal{E} \to \mathcal{R}$ such that $|\mathcal{A}| + |\mathcal{R}| > 2$.

Example: Figure 5.1 illustrates a sample HIN on an academic network, consisting of three node types, i.e., authors(A), papers(P), and conferences (C). Assuming symmetric relations between nodes, we end up with two types of edges, i.e., the author-paper edges, colored in red, and the paper-conference edges, colored in blue.

Definition 5.1.2 (Meta-path)**.** A meta-path $\mathcal{M}$ of length $|\mathcal{M}|$ is defined on the graph network schema as a sequence of the form

$$A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_{|\mathcal{M}|}} A_{|\mathcal{M}|+1},$$

where $A_i \in \mathcal{A}$ and $R_i \in \mathcal{R}$. Using $\circ$ to denote the composition operator on relations, a meta-path can be viewed as a composite relation $R_{|\mathcal{M}|} \circ R_{|\mathcal{M}|-1} \cdots \circ R_1$ between A_1 and $A_{|\mathcal{M}|+1}$. A meta-path is often abbreviated by the sequence of the involved node types i.e. $(A_1 A_2 \cdots A_{|\mathcal{M}|+1})$.

Example: In figure 5.1, although no direct relation exists between authors, we can build semantic relations between them based on the co-authored papers. This results in the meta-path (APA). We can also go a step further and define a longer meta-path (APCPA) by including conference nodes. Similarly, semantics between two papers can be defined in terms of the same authors or same conferences, yielding two possible meta-paths, i.e., (PAP) and (PCP).

In this work, we denote the p-th meta-path by $\mathcal{M}_p$ and the set of the samples of $\mathcal{M}_p$ by $\{m_p\}$ where m_p refers to a single sample of $\mathcal{M}_p$. The node at n-th position in m_p is denoted by $m_p(n)$.

5.2 Related Work

Unsupervised Network Embedding: Most of the earlier work on network embedding targets homogeneous networks and employs random-walk-based objectives, e.g., DeepWalk [63], Node2Vec [75], and LINE [83]. Afterward, the success of different graph neural network (GNN) architectures (e.g., graph convolutional network or GCN [40], graph attention network or GAT [46] and GraphSAGE [48], etc.) gave rise to GNN based network embedding approaches [25, 26]. For instance, graph autoencoders(GAE and VGAE [39]) extend the idea of variational autoencoders [134] to graph datasets while deploying GCN modules to encode latent node embeddings as gaussian random variables. Deep Graph Infomax (DGI [49]) is another interesting approach that employs a GCN encoder. DGI extends the idea of Deep Infomax [103] to the graph domain and learns network embedding in a contrastive fashion by maximizing mutual information between a graph-level representation and high-level node representations. Nonetheless, these methods are restricted to homogeneous graphs and fail to efficiently learn the semantic relations in HINs.

Many HIN embedding methods, for instance Metapath2Vec [132], HIN2Vec [65], RHINE [135] and HERec [133], etc., are partially inspired by homogeneous network embedding techniques in the sense that they employ meta-paths based random walks for learning HIN embedding. In doing so, they inherit the known challenges of random walks such as high dependence on hyperparameters and sacrificing structural information to preserve proximity information [84]. Moreover, their performance highly depends upon the selected meta-path or the strategy adopted to fuse the information from different meta-paths. Recent literature also proposes HIN embedding using approaches that do not depend on one meta-path. For instance, a jump-and-stay strategy is proposed in [64] for learning the HIN embedding. HeGAN [136] employs a generative adversarial approach for HIN embedding. Following DGI [49], HDGI [137] adopts a contrastive approach based on mutual information maximization. NSHE [138] is another HIN-related approach that learns two sets of embeddings to simultaneously preserve information in edges and multiple network schema. DHNE [139] is a hyper-network embedding-based approach that can be used to learn HIN embedding by considering meta-paths instances as hyper-edges.

Community Detection: For homogeneous networks, we classify the approaches for community detection into the following three classes:

1. The most basic approach is the unsupervised learning of the network embedding, followed by a clustering algorithm, e.g., K-Means or GMM, on the embedding.

2. Some architectures learn the network embedding with the primary objective to find good cluster/community assignments for nodes [140, 141]. However, they

usually do not perform well on downstream tasks such as node classification as the embeddings are primarily aimed to find clusters/communities.

3. There are some other approaches, e.g., CommunityGAN [123], CNRL [58] and GEMSEC [122], etc., that take motivation from the euclidean domain to jointly learn the network embedding and the community assignments.

To the best of our knowledge, there exists no approach for HINs that can be classified as (3) as per the above list. For HINs, the known methods use mostly the basic approach, stated in (1), to infer the community assignments from the HIN embedding. At this point, it is worth clarifying that in theory, the output of any hidden layer in a sequential GNN can be viewed as some sort of network embedding. However, these representations are highly task-specific, just like the output of a penultimate layer of a simple convolutional network used for image classification. To address this very concern, *the methods, that claim to learn unsupervised network embedding, evaluate the quality of the embedding on some downstream task* like link prediction or transductive node classification etc., as done by the class-(3) techniques for homogeneous networks such as [58–60, 122] etc. It is worth noticing that devising an approach of class (3) for HINs is non-trivial as it requires explicit modeling of heterogeneity as well as a revision of the sampling techniques. VaCA-HINE is classified as (3) as it presents a unified approach for community-aware HIN embedding that can also be utilized for downstream tasks such as node classification.

5.3 Problem Formulation

Suppose a HIN $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R}, \pi, \lambda\}$ and a matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ of node features, N being the number of nodes. Let $\mathbf{A} \in \{0, 1\}^{N \times N}$ denote the adjacency matrix of $\mathcal{G}$, with a_{ij} referring to the element in i-th row and j-th column of $\mathbf{A}$. Given K as the number of communities, our aim is to jointly learn the d-dimensional community embeddings as well as the HIN embedding such that these embeddings can be used for community detection as well as downstream node classification.

Figure 5.2 gives an overview of the VaCA-HINE framework. It consists of a variational module and a set of contrastive modules. The main goal of the variational module is to optimize the reconstruction of $\mathbf{A}$ by simultaneously incorporating the information in the HIN edges and the communities. The contrastive modules aim to preserve the high-order structure by discriminating between the positive and negative samples of different meta-paths. So, if M meta-paths are selected to preserve high-order HIN structure, the overall loss function, to be minimized, becomes

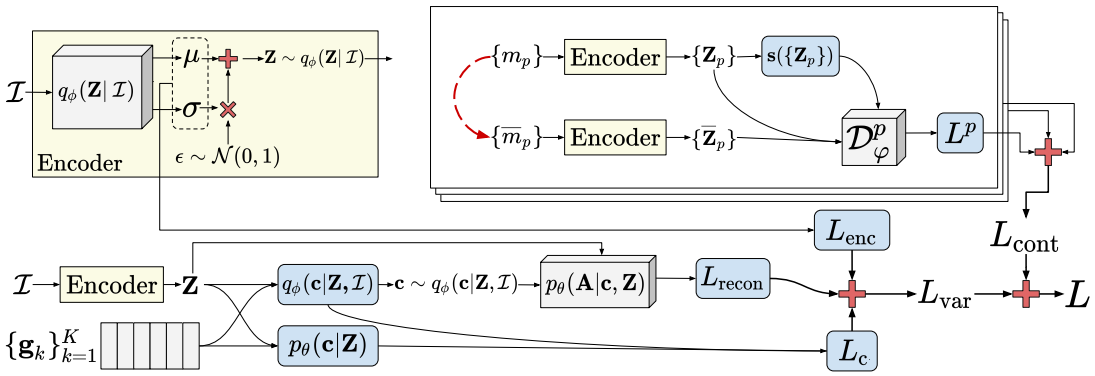


Figure 5.2: Overview of VACA-HINE architecture. For illustrational convenience, the variational encoder block has been extracted to the top-left. All the encoder blocks have been colored yellow to highlight that a single shared encoder is used in the whole architecture. Blocks containing learnable and non-learnable parameters are colored gray and blue respectively. On the bottom, lies the variational module for learning the node embeddings $\mathbf{Z}$ and the community embeddings $\{\mathbf{g}_k\}_{k=1}^K$ such that the variational loss L_{var}, given in equation 5.8, is minimized. The p-th contrastive module lies on the top right. It discriminates between true/positive and corrupted/negative samples for the meta-path $\mathcal{M}_p$. The red-dashed arrow indicates the corruption of the positive samples $\{m_p\}$ of the meta-path $\mathcal{M}_p$, to generate negative samples $\{\overline{m}_p\}$. The encoded representations of the true and corrupted versions of the respective meta-path samples are denoted by $\{\mathbf{Z}_p\}$ and $\{\overline{\mathbf{Z}}_p\}$. These representations, along with the summary vectors $\boldsymbol{s}\big(\{\mathbf{Z}_p\}\big)$, are fed to the discriminator $\mathcal{D}^p_\varphi$ to distinguish between positive and negative samples.

$$L = L_{\text{var}} + L_{\text{cont}} \tag{5.1}$$

$$= L_{\text{var}} + \sum_{p=1}^{M} L^p, \tag{5.2}$$

where L_{var} and L_{cont} refer to the loss of variational module and contrastive modules respectively. We now discuss these modules along with their losses in detail.

5.4 Variational Module

The objective of the variational module is to recover the adjacency matrix $\mathbf{A}$. More precisely, we aim to learn the free parameters θ of our model such that $\log\big(p_\theta(\mathbf{A})\big)$ is maximized. This module is closely related to section 4.2.2 in terms of the underlying objective and basic assumptions on prior $p_\theta(\cdot)$ and the approximate posterior $q_\phi(\cdot)$. So, we can write $\log\big(p_\theta(\mathbf{A})\big)$ as

$$\log\Big(p_\theta(\mathbf{A})\Big) = \log\Bigg(\int \sum_{\mathbf{c}} p_\theta(\mathbf{Z},\mathbf{c},\mathbf{A})d\mathbf{Z}\Bigg) \tag{5.3}$$

$$= \log\Bigg(\int \sum_{\mathbf{c}} p(\mathbf{Z})p_\theta(\mathbf{c}|\mathbf{Z})p_\theta(\mathbf{A}|\mathbf{c},\mathbf{Z})d\mathbf{Z}\Bigg) \tag{5.4}$$

$$= \log\Bigg(\mathbb{E}_{q_\phi(\mathbf{Z},\mathbf{c}|\mathcal{I})}\Bigg\{\frac{p(\mathbf{Z})\ p_\theta(\mathbf{c}|\mathbf{Z})\ p_\theta(\mathbf{A}|\mathbf{c},\mathbf{Z})}{q_\phi(\mathbf{Z}|\mathcal{I})q_\phi(\mathbf{c}|\mathbf{Z},\mathcal{I})}\Bigg\}\Bigg) \tag{5.5}$$

$$\geq \mathbb{E}_{q_\phi(\mathbf{Z},\mathbf{c}|\mathcal{I})}\Bigg\{\log\Bigg(\frac{p(\mathbf{Z})\ p_\theta(\mathbf{c}|\mathbf{Z})\ p_\theta(\mathbf{A}|\mathbf{c},\mathbf{Z})}{q_\phi(\mathbf{Z}|\mathcal{I})q_\phi(\mathbf{c}|\mathbf{Z},\mathcal{I})}\Bigg)\Bigg\}, \tag{5.6}$$

where we define $\mathcal{I} = (\mathbf{A}, \pi, \lambda, \mathbf{X})$ for notational convenience. Here 5.6 follows from Jensen's Inequality. So we maximize, with respect to the parameters θ and ϕ, the corresponding ELBO bound $\mathcal{L}_{\text{ELBO}}$ in 5.6, given by

$$\begin{aligned}\mathcal{L}_{\text{ELBO}} \approx & \underbrace{-\sum_{i=1}^{N} D_{KL}(q_\phi(\mathbf{z}_i|\mathcal{I})\ ||\ p(\mathbf{z}_i))}_{L_{\text{enc}}} \\ & \underbrace{-\sum_{i=1}^{N}\frac{1}{\mathrm{J}}\sum_{j=1}^{\mathrm{J}} D_{KL}(q_\phi(c_i|\mathbf{z}_i^{(j)},\mathcal{I})\ ||\ p_\theta(c_i|\mathbf{z}_i^{(j)}))}_{L_{\text{c}}} \\ & \underbrace{+\sum_{(i,j)\in\mathcal{E}} \mathbb{E}_{q_\phi(\mathbf{z}_i,\mathbf{z}_j,c_i,c_j|\mathcal{I})}\Big\{\log\Big(p_\theta(\mathrm{a}_{ij}|c_i,c_j,\mathbf{z}_i,\mathbf{z}_j)\Big)\Big\}}_{-L_{\text{recon}}},\end{aligned} \tag{5.7}$$

where $D_{KL}(\cdot||\cdot)$ represents the KL-divergence between two distributions. The detailed derivation of equation 5.7 is provided in appendix B. It is worth noticing that Equation 5.7 is similar to equation 4.10. The individual terms $p(\mathbf{z}_i)$, $q_\phi(\mathbf{z}_i|\mathcal{I})$, $p_\theta(c_i|\mathbf{z}_i)$, $q_\phi(c_i|\mathbf{z}_i,\mathcal{I})$, and $p_\theta(\mathrm{a}_{ij}|c_i,c_j,\mathbf{z}_i,\mathbf{z}_j)$ in 5.7 are obtained by applying the same assumptions on the random variables as in section 4.2.2.

Instead of maximizing the ELBO bound, we can minimize the corresponding loss, which we refer to as the variational loss or L_{var}, given by

$$L_{\text{var}} = -\mathcal{L}_{\text{ELBO}} = L_{\text{enc}} + L_{\text{c}} + L_{\text{recon}} \tag{5.8}$$

5.4.1 Choice of Distributions

The distributions involved in L_{enc} and L_{recon} are the same as briefed in section 4.2.3. For the term L_{c}, we need to carefully incorporate the information in the K community embeddings $\{\mathbf{g}_1, \mathbf{g}_2, \cdots, \mathbf{g}_K\}$, $\mathbf{g}_k \in \mathbb{R}^d$ as well as the high order HIN structure.

The prior $p_\theta(c_i|\mathbf{z}_i)$ is the same as equation 4.13. The corresponding approximate posterior $q_\phi(c_i|\mathbf{z}_i, \mathcal{I})$ in equation 5.7 is conditioned on the node embedding $\mathbf{z}_i$ as well as the high-order HIN structure. The intuition governing its design is that if i-th node falls into k-th community, its immediate neighbors as well as high-order neighbors have a relatively higher probability of falling in the k-th community, compared to the other nodes. To model this mathematically, we make use of the samples of the meta-paths involving the i-th node. Let us denote this set as $\zeta_i = \{m \mid \exists n : m(n) = i\}$. For every meta-path sample, we average the embeddings of the nodes constituting the sample. Afterward, we get a single representative embedding by averaging over ζ_i as

$$\hat{\mathbf{z}}_i = \frac{1}{|\zeta_i|} \sum_{m \in \zeta_i} \frac{1}{|m|+1} \sum_{n=1}^{|m|+1} \mathbf{z}_{m(n)}. \tag{5.9}$$

The posterior distribution $q_\phi(c_i|\mathbf{z}_i, \mathcal{I})$ can now be modeled similar to the prior as

$$q_\phi(c_i = k|\mathbf{z}_i, \mathcal{I}) = \text{softmax}(< \hat{\mathbf{z}}_i, \mathbf{g}_k >). \tag{5.10}$$

So, while the prior targets only $\mathbf{z}_i$ for c_i, the equation 5.10 relies on the high-order HIN structure by targeting meta-path-based neighbors. Minimizing the KL-divergence between these two distributions ultimately results in an agreement where nearby nodes tend to have the same community assignments and vice versa.

5.4.2 Practical Considerations

Following the same pattern as in section 4.2.3, we go through the individual loss terms in L_{var} and discuss the practical aspects related to minimization of these terms.

5.4.2.1 Considerations Related to L_{enc}

For computational stability, we learn $\log(\sigma)$ instead of variance in $q_\phi(\mathbf{z}|\mathcal{I})$. In theory, the parameters of $q_\phi(\mathbf{z}_i|\mathcal{I})$ can be learned by any suitable network, e.g., relational graph convolutional network (RGCN [62]), graph convolutional network (GCN [40]), graph attention network (GAT [46]), GraphSAGE [48], or even two linear modules. In practice, some encoders may give better HIN embedding than others. Appendix C.2 contains further discussion on this. For efficient backpropagation, we use the reparameterization trick [19] as illustrated in the encoder block in figure 5.2.

5.4.2.2 Considerations Related to L_{c}

Since the community assignments in equation 5.10 follow a categorical distribution, we use Gumbel-softmax [124] for efficient back-propagation of the gradients. In

the special case where a node i does not appear in any meta-path samples, we formulate $\hat{\mathbf{z}}_i$ in 5.9 by averaging over its immediate neighbors. i.e.

$$\hat{\mathbf{z}}_i = \frac{1}{|\{j : \mathrm{a}_{ij} = 1\}|} \sum_{j:\ \mathrm{a}_{ij}=1} \mathbf{z}_j, \tag{5.11}$$

i.e., when meta-path-based neighbors are not available for a node, we restrict ourselves to leveraging the information in first-order neighbors only.

5.4.2.3 Considerations Related to L_{recon}

The decoder block is the same as defined for the homogeneous networks in section 4.2.4 because it aims to simply discern between positive and negative edges.

5.5 Contrastive Modules

In addition to the variational module, VaCA-HINE architecture consists of M contrastive modules, each referring to a meta-path. All the contrastive modules target the same HIN embedding $\mathbf{Z}$ generated by the variational module. Every module contributes to the enrichment of the HIN embedding by aiming to preserve the high-order HIN structure corresponding to a certain meta-path. This, along with the variational module, enables VaCA-HINE to learn HIN embedding by exploiting the local information (in the form of pairwise relations) as well as the global information (in the form of community assignments and samples of different meta-paths). In this section, we discuss the architecture and the loss of p-th module as shown in the figure 5.2.

5.5.1 Architecture of p-th Contrastive Module

5.5.1.1 Input

The input to p-th module is the set $\{m_p\}$ consisting of the samples of the meta-path $\mathcal{M}_p$.

5.5.1.2 Corruption

As indicated by the red-dashed arrow in figure 5.2, a corruption function is used to generate the set $\{\overline{m}_p\}$ of negative/corrupted samples using the HIN information related to true samples, where $\overline{m}_p$ is a negative sample corresponding to m_p. VaCA-HINE performs corruption by permuting the matrix $\mathbf{X}$ row-wise to shuffle the features between different nodes.

5.5.1.3 Encoder

All the contrastive blocks in VACA-HINE use the same encoder, as used in the variational module. For a sample m_p, the output of encoder is denoted by the matrix $\mathbf{Z}_p \in \mathbb{R}^{(|\mathcal{M}_p|+1)\times d}$ given by

$$\mathbf{Z}_p = [\mathbf{z}_{m_p(1)}, \mathbf{z}_{m_p(2)}, \cdots, \mathbf{z}_{m_p(|\mathcal{M}_p|+1)}] \tag{5.12}$$

Following the same approach, we can obtain $\overline{\mathbf{Z}}_p$ for a corrupted sample $\overline{m}_p$.

5.5.1.4 Summaries

The set of the representations of $\{m_p\}$, denoted by $\{\mathbf{Z}_p\}$, is used to generate the *summary* vectors $\boldsymbol{s}(m_p)$ given by

$$\boldsymbol{s}(m_p) = \frac{1}{|\mathcal{M}_p|+1} \sum_{n=1}^{|\mathcal{M}_p|+1} \mathbf{z}_{m_p(n)}. \tag{5.13}$$

5.5.1.5 Discriminator

The discriminator of p-th block is denoted by $\mathcal{D}^p_\varphi$ where φ collectively refers to the parameters of all the contrastive modules. The aim of $\mathcal{D}^p_\varphi$ is to differentiate between positive and corrupted samples of $\mathcal{M}_p$. It takes the representations $\{\mathbf{Z}_p\}$ and $\{\overline{\mathbf{Z}}_p\}$ along with the summary vectors at the input, and outputs a binary decision for every sample, i.e., 1 for positive and 0 for negative or corrupted samples. Specifically, for a sample m_p, the discriminator $\mathcal{D}^p_\varphi$ first flattens $\mathbf{Z}_p$ in equation 5.12 into $\boldsymbol{v}_p$ by vertically stacking the columns of $\mathbf{Z}_p$ as

$$\boldsymbol{v}_p = \Big\Vert_{n=1}^{|\mathcal{M}_p|+1} \mathbf{z}_{m_p(n)}, \tag{5.14}$$

where $\|$ denotes the concatenation operator. Afterward, the output of $\mathcal{D}^p_\varphi$ is given by

$$\mathcal{D}^p_\varphi\Big(m_p, \boldsymbol{s}(m_p)\Big) = \text{sigmoid}\Big(\boldsymbol{v}_p^T \boldsymbol{W}_p \boldsymbol{s}(m_p)\Big), \tag{5.15}$$

where $\boldsymbol{W}_p \in \mathbb{R}^{(|\mathcal{M}_p|+1)\times d}$ is the learnable weight matrix of $\mathcal{D}^p_\varphi$. Similarly, for negative sample $\overline{m}_p$,

$$\mathcal{D}^p_\varphi\Big(\overline{m}_p, \boldsymbol{s}(m_p)\Big) = \text{sigmoid}\Big(\overline{\boldsymbol{v}}_p^T \boldsymbol{W}_p \boldsymbol{s}(m_p)\Big), \tag{5.16}$$

where $\overline{\boldsymbol{v}}_p^T$ is the flattened version of $\overline{\mathbf{Z}}_p$.

5.5.2 Loss of p-th Contrastive Module

The outputs in equation 5.15 and 5.16 can be respectively viewed as the probabilities of m_p and $\overline{m}_p$ being positive samples according to the discriminator. Consequently, we can model the contrastive loss L^p, associated with the p-th block, in terms of the binary cross-entropy loss between the positive and the corrupted samples. Considering J negative samples $\overline{m}_p^{(j)}$ for every positive sample m_p, the loss L^p can be written as

$$L^p = -\frac{1}{(J+1) \times |\{m_p\}|} \sum_{m_p \in \{m_p\}} \left(\log\Big[\mathcal{D}_\varphi^p\Big(m_p, \boldsymbol{s}(m_p)\Big)\Big] + \sum_{j=1}^{J} \log\Big[1 - \mathcal{D}_\varphi^p\Big(\overline{m}_p^{(j)}, \boldsymbol{s}(m_p)\Big)\Big] \right). \quad (5.17)$$

Minimizing L^p preserves the high-order HIN structure by ensuring that the correct samples of $\mathcal{M}_p$ are distinguishable from the corrupted ones. This module can be replicated for different meta-paths. Hence, given M meta-paths, we have M contrastive modules, each one aiming to preserve the integrity of the samples of a specific meta-path by using a separate discriminator. The resulting loss, denoted by L_{cont}, is the sum of the losses from all the contrastive modules as defined in equation 5.2. Since the encoder is shared between the variational module and the contrastive modules, minimizing L_{cont} ensures that the node embeddings also leverage the information in the high-order HIN structure.

5.6 Experiments

In this section, we conduct an extensive empirical evaluation of our approach. We start with a brief description of the datasets and the baselines selected for evaluation. Afterward, we provide the experimental setup for the baselines as well as for VaCA-HINE . Then we analyze the results of community detection and downstream node classification tasks. Since most of our competitors limit themselves to non-overlapping communities, we follow the same for sake of fair comparison. Furthermore, appendix C.2 contains the evaluation with different encoder modules both with and without contrastive loss L_{cont}.

5.6.1 Datasets

We select two academic networks and a subset of IMDB [142] for our experiments. The detailed statistics of these datasets are given in table 5.1.

DBLP [62]: The extracted subset of DBLP has the author and paper nodes divided into four areas, i.e., database, data mining, machine learning, and information retrieval. We report the results of community detection and node classification for author nodes as well as paper nodes by comparing with the research area as the ground truth. We do not report the results for conference nodes as they are

Name	Nodes	#Nodes	Relations	#Relations	Meta-paths
DBLP	Papers (P) Authors (A) Conferences (C)	9556 2000 20	P-A P-C	18304 9556	ACPCA APA ACA
ACM	Papers (P) Authors (A) Subjects (S)	4019 7167 60	P-A P-S	13407 4019	PAP PSP
IMDB	Movies (M) Actors (A) Directors (D)	3676 4353 1678	M-A M-D	11028 3676	MAM MDM

Table 5.1: Statistics of the datasets used for evaluation.

only 20 in number. To distinguish between the results of different node types, we use the names DBLP-A and DBLP-P respectively for the part of DBLP with the node-type authors and papers.

ACM [142]: Following [142] and [138], the extracted subset of ACM is chosen with papers published in KDD, SIGMOD, SIGCOMM, MobiCOMM and VLDB. The paper features are the bag-of-words representation of keywords. They are classified based on whether they belong to data mining, database, or wireless communication.

IMDB [142]: Here we use the extracted subset of movies classified according to their genre, i.e., action, comedy, and drama.

5.6.2 Baselines

We compare the performance of VaCA-HINE with 15 competitive baselines used for unsupervised network embedding. These baselines include 5 homogeneous network embedding methods, 3 techniques proposed for joint homogeneous network embedding and community detection, and 7 HIN embedding approaches. **DeepWalk [63]** learns the node embeddings by first performing classical truncated random walks on an input graph, followed by the skip-gram model. **LINE [83]** samples node pairs directly from a homogeneous graph and learns node embeddings with the aim of preserving first-order or second-order proximity, respectively denoted by LINE-1 and LINE-2 in this work. **GAE [39]** extends the idea of autoencoders to graph datasets. The aim here is to reconstruct $\mathbf{A}$ for the input homogeneous graph. **VGAE [39]** is the variational counterpart of GAE. It models the latent node embeddings as Gaussian random variables and aims to optimize $\log\big(p(\mathbf{A})\big)$ using a variational model. **DGI [49]** extends Deep-Infomax [103] to graphs. This is a contrastive approach to learning network embedding such that the true samples share a higher similarity to a global network representation (known as *summary*), as compared to the corrupted samples. **GEMSEC [122]** jointly learns network embedding and node clustering assignments by sequence sampling. **CNRL [58]** makes use of the node sequences generated by random-walk-based techniques, to

jointly learn node communities and node embeddings. **CommunityGAN [123]** is a generative adversarial approach for learning community-based network embedding. Given K as the number of desired communities, it learns K-dimensional latent node embeddings such that every dimension gives the assignment strength for the target node in a certain community. **Metapath2Vec [132]** is a HIN embedding approach that makes use of a meta-path when performing random walks on graphs. The generated random walks are then fed to a heterogeneous skip-gram model, thus preserving the semantics-based similarities in a HIN. **HIN2Vec [65]** jointly learns the embeddings of nodes as well as meta-paths, thus preserving the HIN semantics. The node embeddings are learned such that they can predict the meta-paths connecting them. **HERec [133]** learns semantic-preserving HIN embedding by designing a type-constraint strategy for filtering the node-sequences based upon meta-paths. Afterward, skip-gram model is applied to get the HIN embedding. **HDGI [137]** extends DGI model to HINs. The main idea is to disassemble a HIN into multiple homogeneous graphs based upon different meta-paths, followed by semantic-level attention to aggregate different node representations. Afterward, a contrastive approach is applied to maximize the mutual information between the high-level node representations and the graph representation. **DHNE [139]** learns hyper-network embedding by modeling the relations between the nodes in terms of indecomposable hyper-edges. It uses deep autoencoders and classifiers to realize a non-linear tuple-wise similarity function while preserving both local and global proximities in the formed embedding space. **NSHE [138]** embeds a HIN by jointly learning two embeddings: one for optimizing the pairwise proximity, as dictated by $\mathbf{A}$, and the second one for preserving the high-order proximity, as dictated by the meta-path samples. **HeGAN [136]** is a generative adversarial approach for learning HIN embedding by discriminating between real and fake heterogeneous relations.

5.6.3 Implementation Details

For Baselines: The embedding dimension d for all the methods is kept to 128. The only exception is CommunityGAN because it requires the node embeddings to have the same dimensions as the desired communities, i.e., $d = K$. The rest of the parameters for all the competitors are kept the same as given by their authors. For GAE and VGAE, the number of sampled negative edges is the same as the positive edges as given in the original implementations. For the competitors that are designed for homogeneous graphs (i.e., DeepWalk, LINE, GAE, VGAE, and DGI), we treat the HINs as homogeneous graphs. For the methods Metapath2Vec and HERec, we test all the meta-paths in table 5.1 and report the best results. For DHNE, the meta-path instances are considered as hyper-edges. The embeddings obtained by DeepWalk are used for all the models that require node features. **For VaCA-HINE :** The meta-path samples are generated using dgl [143] and the overall architecture is implemented in pytorch-geometric [144]. For every meta-path,

we select 10 samples from every starting node. For the loss L^p in equation 5.17, we select $J = 1$, i.e., we generate one negative sample for every positive sample. The joint community assignments are obtained as the `argmax` of the corresponding community distribution. All the results are presented as an average of 10 runs. Kindly refer to [145] and the appendix C.1 for further details of implementation.

5.6.4 Community Detection

We start by evaluating the learned embeddings on community detection. Apart from GEMSEC, CNRL, and CommunityGAN, no baseline learns node cluster/community assignments jointly with the embeddings. Therefore we choose K-Means to find the community assignments for such algorithms. For VaCA-HINE we give the results both with and without L_{cont}. Moreover, for comparison with the baselines, we also look at the case where the community embeddings $\mathbf{g}_k$ are used during joint training but not used for extracting the community assignments. Instead, we use the assignments obtained by directly fitting K-Means on the learned HIN embedding. We use normalized mutual information (NMI) score for the quantitative evaluation of the performance.

Table 5.2 gives the comparison of different algorithms for community detection. The results in the table are divided into three sections i.e., the approaches dealing with homogeneous networks, the approaches for unsupervised HIN embedding, and different variants of the proposed method. VaCA-HINE results are the best on all the datasets. In addition, the use of L_{cont} improves the results in 3 out of 4 cases, the exception being the IMDB dataset. It is rather difficult to comment on the reason behind this because this dataset is inherently not easy to cluster as demonstrated by the low NMI scores for all the algorithms. Whether or not we use L_{cont}, the performance achieved by using the jointly learned community assignments is always better than the one where K-Means is used to cluster the HIN embedding. So joint learning of community assignments and network embedding consistently outperforms the case where these two tasks are treated independently. This conforms to the results obtained in the domain of euclidean data and homogeneous networks. It also highlights the efficacy of the variational module and validates the intuition behind the choices of involved distributions. Moreover, as shown by the last two rows of table 5.2, L_{cont} has a negative effect on the performance in 3 out of 4 datasets if K-Means is used to get community assignments. Apart from VaCA-HINE , the approaches that explicitly model HIN structure (e.g., HeGAN, NSHE, and Metapath2Vec) generally perform better on HINs.

5.6.5 Transductive Node Classification

For node classification, we first learn the HIN embedding in an unsupervised manner for each of the four cases. Afterward, 80% of the labels are used for training the logistic classifier using `lbfgs` solver [74, 146]. We keep the split the

Method	DBLP-P	DBLP-A	ACM	IMDB
DeepWalk	46.75	66.25	48.81	0.41
LINE-1	42.18	29.98	37.75	0.03
LINE-2	46.83	61.11	41.8	0.03
GAE	63.21	65.43	41.03	2.91
VGAE	62.76	63.42	42.14	3.51
DGI	37.33	10.98	39.66	0.53
GEMSEC	41.18	62.22	32.69	0.21
CNRL	39.02	66.18	36.81	0.30
CommunityGAN	41.43	66.93	38.06	0.68
DHNE	35.33	21.00	20.25	0.05
Metapath2Vec	56.89	68.74	42.71	0.09
HIN2Vec	30.47	65.79	42.28	0.04
HERec	39.46	24.09	40.70	0.51
HDGI	41.48	29.46	41.05	0.71
NSHE	65.54	69.52	42.19	5.61
HeGAN	60.78	68.95	43.35	6.56
VaCA-HINE with $\mathbf{g}_k$ and L_{cont}	**72.85**	**71.25**	**52.44**	6.63
VaCA-HINE with $\mathbf{g}_k$ without L_{cont}	72.35	69.85	50.65	**7.59**
VaCA-HINE with KM and L_{cont}	70.33	68.30	50.93	4.58
VaCA-HINE with KM without L_{cont}	70.89	69.47	51.77	3.38

Table 5.2: NMI scores on community detection task. Best results for each dataset are bold. For comparison, we give the results both with and without contrastive loss L_{cont}. For VaCA-HINE results, We use $\mathbf{g}_k$ when the results are obtained by utilizing community embeddings for inferring the community assignments. Moreover, we also give the results obtained by fitting K-Means (abbreviated as KM) on the learned embeddings.

same as our competitors for a fair comparison. The performance is evaluated using F1-Micro and F1-macro scores. Table 5.3 gives a comparison of VaCA-HINE with the baselines. In all the datasets, VaCA-HINE gives the best performance. However, the performance suffers when L_{cont} is ignored, thereby providing empirical evidence of the utility of the contrastive modules for improving HIN embedding quality. The improvement margin is maximum in the case of DBLP-P because relatively fewer labeled nodes are available for this dataset. So, compared to the competitors, correctly classifying even a few more nodes enables us to achieve perfect results. Among the methods that jointly learn homogeneous network embedding and community assignments, CommunityGAN performs poorly in particular. A possible reason is restricting the latent dimensions to be the same as the number of communities. While it can yield acceptable community assignments, it makes the downstream node classification difficult for *linear* classifiers, especially

Method	F1-Micro / F1-Macro			
	DBLP-P	**DBLP-A**	**ACM**	**IMDB**
DeepWalk	90.12 / 89.45	89.44 / 88.48	82.17 / 81.82	56.52 / 55.24
LINE-1	81.43 / 80.74	82.32 / 80.2	82.46 / 82.35	43.75 / 39.87
LINE-2	84.76 / 83.45	88.76 / 87.35	82.21 / 81.32	40.54 / 33.06
GAE	90.09 / 89.23	77.54 / 60.21	81.97 / 81.47	55.16 / 53.57
VGAE	90.95 / 90.04	71.43 / 69.03	81.59 / 81.30	57.47 / 56.06
DGI	95.24 / 94.51	91.81 / 91.20	81.65 / 81.79	58.42 / 56.94
GEMSEC	78.38 / 77.85	87.45 / 87.01	62.05 / 61.24	51.56 / 50.2
CNRL	81.43 / 80.83	83.20 / 82.68	66.58 / 65.28	43.12 / 41.48
CommunityGAN	72.01 / 66.54	67.81 / 71.55	52.87 / 51.67	33.81 / 31.09
DHNE	85.71 / 84.67	73.30 / 67.61	65.27 / 62.31	38.99 / 30.53
Metapath2Vec	92.86 / 92.44	89.36 / 87.95	83.60 / 82.77	51.90 / 50.21
HIN2Vec	83.81 / 83.85	90.30 / 89.46	54.30 / 48.59	48.02 / 46.24
HERec	90.47 / 87.50	86.21 / 84.55	81.89 / 81.74	54.48 / 53.46
HDGI	95.24 / 94.51	92.27 / 91.81	82.06 / 81.64	57.81 / 55.73
NSHE	95.24 / 94.76	93.10 / 92.37	82.52 / 82.67	59.21 / 58.31
HeGAN	88.79 / 83.81	90.48 / 89.27	83.09 / 82.94	58.56 / 57.12
VaCA-HINE	**100.00 / 100.00**	**93.57 / 92.80**	**83.70 / 83.48**	**60.73 / 59.28**
VaCA-HINE without L_{cont}	**100.00 / 100.00**	93.10 / 92.22	80.46 / 79.85	57.60 / 53.27

Table 5.3: F1-Micro and F1-Macro scores on node classification task. Best results for each dataset are bold. For comparison, we give the results both with and without contrastive loss L_{cont}. It can be readily observed that VaCA-HINE outperforms the baselines for all the datasets. Moreover, the results are always improved by including the contrastive modules.

when K is small. An interesting observation lies in the results of contrastive approaches (DGI and HDGI) and autoencoder-based models in table 5.2 and table 5.3. For community detection in table 5.2, DGI/HDGI results are usually quite poor compared to GAE and VGAE. However, for classification in table 5.3, DGI and HDGI almost always outperform GAE and VGAE. This gives a hint that an approach based on edge reconstruction might be better suited for HIN community detection, whereas a contrastive approach could help in the general improvement of node embedding quality as evaluated by the downstream node classification. Since VaCA-HINE makes use of a variational module (aimed to reconstruct $\mathbf{A}$) as well as the contrastive modules, it performs well in both tasks.

5.7 Conclusion

We make the first attempt at joint learning of community assignments and HIN embedding. This is achieved by refining a single target HIN embedding using a variational module and multiple contrastive modules. The variational module aims at the reconstruction of the adjacency matrix $\mathbf{A}$ in a community-aware manner. The contrastive modules attempt to preserve the high-order HIN structure. Specifically, every contrastive module attempts to distinguish between positive and negative/corrupted samples of a certain meta-path. The joint training of the variational and contrastive modules yields the HIN embedding that leverages the local information (provided by the pairwise relations) as well as the global information (present in community assignments and high-order semantics as dictated by the samples of different meta-paths). In addition to the HIN embedding, VaCA-HINE simultaneously learns the community embeddings and consequently the community assignments for HIN nodes. These jointly learned community assignments consistently outperform many many competitive baselines in community detection as well as downstream node classification task.

6 Conclusion and Outlook

In this work, we have studied the framework of the Variational Autoencoder (VAE) for unsupervised network embedding in graph datasets. We have focused on the datasets which exhibit homophily, and attempted to extend VAE architecture for general network embedding as well as community-aware network embedding. We start with the analysis of over-pruning in VGAE - the extension of VAE proposed by Kipf and Welling [39] - and propose a model-based solution to this problem. Afterward, we attempt to explicitly preserve the information in the immediate as well as the broader neighborhood by exploiting the redundancy-reduction-principle to efficiently fuse the representations of various meta paths.

The underlying assumption of homophily raises an interesting question regarding the latent embeddings $\boldsymbol{z}$: Can the current prior of i.i.d. standard Gaussian be replaced with a better one? For vector datasets, VAE assumes the latent embedding to have $\mathcal{N}(\mathbf{0}, \mathbf{1})$ distribution due to the absence of any additional information. However, for the homophilic graphs, we can exploit the information in the links to build a more expressive prior. For instance, it is reasonable to assume that the latent representation of a node i is the mean of its neighbors and the standard deviation is a scalar multiple of the overall similarity of i with its neighbors. Such a prior could help in a better and more targeted regularization in homophilic graphs which could consequently lead to better representation without manually decreasing the effect of regularization as done in [39, 72]. This is a future work for me to explore.

The second half of this thesis proposes methodologies to jointly learn community embeddings as well as community-aware network embedding. We ensure that the proposed approach is computationally cheaper than its direct competitors. The extension of this approach from homogeneous to heterogeneous networks requires special care, especially when dealing with sampling techniques. To alleviate this issue, we propose embedding the information from all the selected meta-paths in a contrastive manner. This enriches the network embedding with local information from the immediate neighborhoods, as well as global information from community assignments and larger neighborhoods. For all the proposed approaches, we provide an extensive comparison with the direct competitors to demonstrate that all our approaches are on par with the state-of-the-art methodologies, often outperforming them. However, there are two important factors to be kept in mind when using such approaches. The first one is the fact that these approaches need the number of communities to be known beforehand - a requirement that is often not fulfilled in many practical scenarios. The second point to remember is that these approaches

scale with the number of communities. J-Enc and VaCA-HINE are still better than many competitors that scale quadratically with the number of communities. Nonetheless, for networks where the number of communities becomes comparable with the number of nodes, such algorithms can become computationally expensive.

Bibliography

[1] I. H. Sarker. Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2(6):1–20, 2021.

[2] J. Gantz and D. Reinsel. The digital universe decade–are you ready (2010). *URL: http://www. emc. com/collateral/analyst-reports/idcdigital-universe-are-you-ready. pdf*, 2012.

[3] S. Kim, P. A. Thiessen, E. E. Bolton, J. Chen, G. Fu, A. Gindulyte, L. Han, J. He, S. He, B. A. Shoemaker, et al. Pubchem substance and compound databases. *Nucleic acids research*, 44(D1):D1202–D1213, 2016.

[4] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke. The rise of deep learning in drug discovery. *Drug discovery today*, 23(6):1241–1250, 2018.

[5] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha. Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE access*, 7:65579–65615, 2019.

[6] X. Liu, H. Shin, and A. C. Burns. Examining the impact of luxury brand's social media marketing on customer engagement: Using big data analytics and natural language processing. *Journal of Business Research*, 125:815–826, 2021.

[7] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[8] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *Proceedings. international conference on image processing*, volume 1, pages I–I. IEEE, 2002.

[9] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[12] W. Wang, V. W. Zheng, H. Yu, and C. Miao. A survey of zero-shot learning: Settings, methods, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–37, 2019.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[14] D. W. Otter, J. R. Medina, and J. K. Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

[15] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[16] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[17] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR, 2016.

[18] S. Qi, X. Ning, G. Yang, L. Zhang, P. Long, W. Cai, and W. Li. Review of multi-view 3d object recognition methods based on deep learning. *Displays*, 69:102053, 2021.

[19] C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[20] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[21] S. K. Ata, Y. Fang, M. Wu, X.-L. Li, and X. Xiao. Disease gene classification with metagraph representations. *Methods*, 131:83–92, 2017.

[22] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

[23] Z.-X. Wu, X.-J. Xu, Y. Chen, and Y.-H. Wang. Spatial prisoner's dilemma game with volunteering in newman-watts small-world networks. *Physical Review E*, 71(3):037103, 2005.

[24] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[25] H. Cai, V. W. Zheng, and K. C.-C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[26] P. Cui, X. Wang, J. Pei, and W. Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):833–852, 2018.

[27] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[28] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

[29] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[30] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[31] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[32] I. Robinson, J. Webber, and E. Eifrem. *Graph databases: new opportunities for connected data*. " O'Reilly Media, Inc.", 2015.

[33] R. Mihalcea and D. Radev. *Graph-based natural language processing and information retrieval*. Cambridge university press, 2011.

[34] V. Nastase, R. Mihalcea, and D. R. Radev. A survey of graphs in natural language processing. *Natural Language Engineering*, 21(5):665–698, 2015.

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

[36] M. Gallivan and M. Ahuja. Co-authorship, homophily, and scholarly influence in information systems research. *Journal of the Association for Information Systems*, 16(12):2, 2015.

[37] W. A. Brechwald and M. J. Prinstein. Beyond homophily: A decade of advances in understanding peer influence processes. *Journal of research on adolescence*, 21(1):166–179, 2011.

[38] Y. Bu, J. Parkinson, and P. Thaichon. Influencer marketing: Homophily, customer value co-creation behaviour and purchase intention. *Journal of Retailing and Consumer Services*, 66:102904, 2022.

[39] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[40] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. 2016. `arXiv:1609.02907`.

[41] Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[42] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[43] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. How to train deep variational autoencoders and probabilistic ladder networks. In *33rd International Conference on Machine Learning (ICML 2016)*, 2016.

[44] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751, 2016.

[45] S. Yeung, A. Kannan, Y. Dauphin, and L. Fei-Fei. Tackling over-pruning in variational autoencoders. *arXiv preprint arXiv:1706.03643*, 2017.

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

[47] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[48] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[49] P. Velickovic, W. Fedus, W. L. Hamilton, P. Li, Y. Bengio, and R. D. Hjelm. Deep graph infomax. 2019.

[50] 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 *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1150–1160, 2020.

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

[52] H. B. Barlow et al. Possible principles underlying the transformation of sensory messages. *Sensory communication*, 1(01), 1961.

[53] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow twins: Self-supervised learning via redundancy reduction. *arXiv preprint arXiv:2103.03230*, 2021.

[54] J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487. PMLR, 2016.

[55] P. Huang, Y. Huang, W. Wang, and L. Wang. Deep embedding network for clustering. In *2014 22nd International conference on pattern recognition*, pages 1532–1537. IEEE, 2014.

[56] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

[57] N. Dilokthanakul, P. A. Mediano, M. Garnelo, M. C. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

[58] C. Tu, X. Zeng, H. Wang, Z. Zhang, Z. Liu, M. Sun, B. Zhang, and L. Lin. A unified framework for community detection and network representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 31(6):1051–1065, 2018.

[59] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria. Learning community embedding with community detection and node embedding on graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 377–386, 2017.

[60] F.-Y. Sun, M. Qu, J. Hoffmann, C.-W. Huang, and J. Tang. vgraph: A generative model for joint community detection and node representation

learning. In *Advances in Neural Information Processing Systems*, pages 514–524, 2019.

[61] C. Shi, Y. Li, J. Zhang, Y. Sun, and S. Y. Philip. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37, 2016.

[62] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[63] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[64] R. Hussein, D. Yang, and P. Cudré-Mauroux. Are meta-paths necessary? revisiting heterogeneous graph embeddings. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 437–446, 2018.

[65] T.-y. Fu, W.-C. Lee, and Z. Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1797–1806, 2017.

[66] Z. Huang and N. Mamoulis. Heterogeneous information network embedding for meta path based proximity. *arXiv preprint arXiv:1701.05291*, 2017.

[67] T. Chen and Y. Sun. Task-guided and path-augmented heterogeneous network embedding for author identification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 295–304, 2017.

[68] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[69] E. W. Weisstein. Jensen's inequality. From MathWorld—A Wolfram Web Resource. URL: `https://mathworld.wolfram.com/JensensInequality.html`.

[70] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *Iclr*, 2(5):6, 2017.

[71] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner. Understanding disentangling in beta-vae. *arXiv preprint arXiv:1804.03599*, 2018.

[72] T. Kipf. gae. `https://github.com/tkipf/gae/issues/20#issuecomment-446260981`. URL: `https://github.com/tkipf/gae/issues/20#issuecomment-446260981`.

[73] L. Tang and H. Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[75] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[76] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[77] R. A. Khan. Evgae. `https://github.com/RayyanRiaz/EVGAE`, 2020.

[78] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

[79] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. Community preserving network embedding. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[80] J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. *Advances in Neural Information Processing Systems*, 32:13354–13366, 2019.

[81] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[82] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114, 2016.

[83] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[84] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 385–394, 2017.

[85] J. Park, M. Lee, H. J. Chang, K. Lee, and J. Y. Choi. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6519–6528, 2019.

[86] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. Adversarially regularized graph autoencoder for graph embedding. *arXiv preprint arXiv:1802.04407*, 2018.

[87] S. Zheng, Z. Zhu, X. Zhang, Z. Liu, J. Cheng, and Y. Zhao. Distribution-induced bidirectional generative adversarial network for graph representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7224–7233, 2020.

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

[89] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[90] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[91] P. Bielak, T. Kajdanowicz, and N. V. Chawla. Graph barlow twins: A self-supervised representation learning framework for graphs. *Knowledge-Based Systems*, page 109631, 2022.

[92] B. Karrer and M. E. Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

[93] R. A. Khan, M. U. Anwaar, and M. Kleinsteuber. Epitomic variational graph autoencoder, 2020. `arXiv:2004.01468`.

[94] B. Jiang, D. Lin, J. Tang, and B. Luo. Data representation and learning with graph diffusion-embedding networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10414–10423, 2019.

[95] B. Xu, H. Shen, Q. Cao, K. Cen, and X. Cheng. Graph convolutional networks using heat kernel for semi-supervised learning. *arXiv preprint arXiv:2007.16002*, 2020.

[96] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[97] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006.

[98] Z. Wei, X. He, X. Xiao, S. Wang, S. Shang, and J.-R. Wen. Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. In *Proceedings of the 2018 International Conference on Management of Data*, pages 441–456, 2018.

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

[100] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015.

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

[102] R. A. Khan. Implementation code for bvgae. `https://github.com/RayyanRiaz/bvgae`, December 2022.

[103] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[104] C. Mavromatis and G. Karypis. Graph infoclust: Leveraging cluster-level node information for unsupervised graph representation learning. *arXiv preprint arXiv:2009.06946*, 2020.

[105] K. Ma, H. Yang, H. Yang, T. Jin, P. Chen, Y. Chen, B. F. Kamhoua, and J. Cheng. Improving graph representation learning by contrastive regularization. *arXiv preprint arXiv:2101.11525*, 2021.

[106] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pages 2069–2080, 2021.

[107] S. Thakoor, C. Tallec, M. G. Azar, M. Azabou, E. L. Dyer, R. Munos, P. Veličković, and M. Valko. Large-scale representation learning on graphs via bootstrapping. *arXiv preprint arXiv:2102.06514*, 2021.

[108] Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[109] R. A. Khan, M. U. Anwaar, O. Kaddah, Z. Han, and M. Kleinsteuber. Unsupervised learning of joint embeddings for node representation and community detection. In *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 19–35, Cham, 2021. Springer International Publishing.

[110] J. Tang, C. Aggarwal, and H. Liu. Node classification in signed social networks. In *Proceedings of the 2016 SIAM international conference on data mining*, pages 54–62. SIAM, 2016.

[111] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234, 2016.

[112] S. Gao, L. Denoyer, and P. Gallinari. Temporal link prediction by integrating content and structure information. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1169–1174, 2011.

[113] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *nature*, 466(7307):761–764, 2010.

[114] I. Derényi, G. Palla, and T. Vicsek. Clique percolation in random networks. *Physical review letters*, 94(16):160202, 2005.

[115] P. K. Gopalan and D. M. Blei. Efficient discovery of overlapping communities in massive networks. *Proceedings of the National Academy of Sciences*, 110(36):14534–14539, 2013.

[116] J. Leskovec and J. J. Mcauley. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pages 539–547, 2012.

[117] J. Yang, J. McAuley, and J. Leskovec. Community detection in networks with node attributes. In *2013 IEEE 13th International Conference on Data Mining*, pages 1151–1156. IEEE, 2013.

[118] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596, 2013.

[119] M. Kozdoba and S. Mannor. Community detection via measure space embedding. In *Proceedings of the 28th International Conference on Neural*

Information Processing Systems - Volume 2, NIPS'15, pages 2890–2898, Cambridge, MA, USA, 2015. MIT Press.

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

[121] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *Acm computing surveys (csur)*, 45(4):1–35, 2013.

[122] B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton. Gemsec: Graph embedding with self clustering. In *Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining*, pages 65–72, 2019.

[123] Y. Jia, Q. Zhang, W. Zhang, and X. Wang. Communitygan: Community detection with generative adversarial nets. In *The World Wide Web Conference*, pages 784–794, 2019.

[124] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[125] L. Yang, X. Cao, D. He, C. Wang, X. Wang, and W. Zhang. Modularity based community detection with deep learning. In *IJCAI*, volume 16, pages 2252–2258, 2016.

[126] A. Bojchevski and S. Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

[127] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

[128] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[129] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[130] R. A. Khan. J-enc implementation. `https://github.com/RayyanRiaz/gnn_comm_det`, 2021.

[131] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[132] Y. Dong, N. V. Chawla, and A. Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144, 2017.

[133] C. Shi, B. Hu, W. X. Zhao, and S. Y. Philip. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 31(2):357–370, 2018.

[134] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[135] Y. Lu, C. Shi, L. Hu, and Z. Liu. Relation structure-aware heterogeneous information network embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4456–4463, 2019.

[136] B. Hu, Y. Fang, and C. Shi. Adversarial learning on heterogeneous information networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 120–129, 2019.

[137] Y. Ren, B. Liu, C. Huang, P. Dai, L. Bo, and J. Zhang. Heterogeneous deep graph infomax. *arXiv preprint arXiv:1911.08538*, 2019.

[138] J. Zhao, X. Wang, C. Shi, Z. Liu, and Y. Ye. Network schema preserving heterogeneous information network embedding. IJCAI, 2020.

[139] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu. Structural deep embedding for hyper-networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[140] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu. Learning deep representations for graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[141] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020.

[142] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032, 2019.

[143] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[144] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[145] R. A. Khan. Implementation code for vaca-hine. `https://github.com/RayyanRiaz/vacahine`, May 2021.

[146] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 1997.

[147] I. Chami, Z. Ying, C. Ré, and J. Leskovec. Hyperbolic graph convolutional neural networks. *Advances in neural information processing systems*, 32:4868–4879, 2019.

[148] C. Gulcehre, M. Denil, M. Malinowski, A. Razavi, R. Pascanu, K. M. Hermann, P. Battaglia, V. Bapst, D. Raposo, A. Santoro, et al. Hyperbolic attention networks. *arXiv preprint arXiv:1805.09786*, 2018.

[149] N. Mehta, L. C. Duke, and P. Rai. Stochastic blockmodels meet graph neural networks. In *International Conference on Machine Learning*, pages 4466–4474. PMLR, 2019.

[150] A. Grover, A. Zweig, and S. Ermon. Graphite: Iterative generative modeling of graphs. In *International conference on machine learning*, pages 2434–2444. PMLR, 2019.

[151] S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[152] R. Xia, Y. Pan, L. Du, and J. Yin. Robust multi-view spectral clustering via low-rank and sparse decomposition. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28, 2014.

[153] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang. Network representation learning with rich text information. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[154] X. Zhang, H. Liu, Q. Li, and X.-M. Wu. Attributed graph clustering via adaptive graph convolution. *arXiv preprint arXiv:1906.01210*, 2019.

[155] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, and C. Zhang. Attributed graph clustering: A deep attentional embedding approach. *arXiv preprint arXiv:1906.06532*, 2019.

[156] M. Qu, Y. Bengio, and J. Tang. Gmnn: Graph markov neural networks. In *International conference on machine learning*, pages 5241–5250. PMLR, 2019.

A BGAE Supplementary Material

A.1 Derivation of L_{recon} for Variational Case

As mentioned in the paper, we aim to learn the model parameters θ to maximize the log probability of recovering the joint probability of $\mathbf{A}$ and $\mathbf{S}$ from $\mathbf{Z}$, given as

$$\log\Big(p_\theta(\mathbf{A},\mathbf{S})\Big) = \log\Big(\int p_\theta(\mathbf{A},\mathbf{S},\mathbf{Z})d\mathbf{Z}\Big) \tag{A.1}$$

$$= \log\Big(\int p(\mathbf{Z})p_\theta(\mathbf{A}|\mathbf{Z})p_\theta(\mathbf{S}|\mathbf{Z})d\mathbf{Z}\Big). \tag{A.2}$$

Here we assume conditional independence between $\mathbf{A}$ and $\mathbf{S}$ given $\mathbf{Z}$. The approximate posterior, introduced for tractability, is given as

$$q_\phi(\mathbf{Z}|\mathcal{I},\overline{\mathcal{I}}) = q_\phi(\mathbf{Z}^{\mathcal{I}}\mathbf{Z}^{\overline{\mathcal{I}}}|\mathcal{I},\overline{\mathcal{I}}) \tag{A.3}$$

$$= q_\phi(\mathbf{Z}^{\mathcal{I}}|\mathcal{I})q_\phi(\mathbf{Z}^{\overline{\mathcal{I}}}|\mathcal{I},\mathbf{Z}^{\mathcal{I}}) \tag{A.4}$$

$$= q_\phi(\mathbf{Z}^{\mathcal{I}}|\mathcal{I})q_\phi(\mathbf{Z}^{\overline{\mathcal{I}}}|\overline{\mathcal{I}}), \tag{A.5}$$

where equation A.5 follows from equation A.4 because of the assumed conditional independence of $\mathbf{Z}^{\mathcal{I}}$ and $\mathbf{Z}^{\overline{\mathcal{I}}}$ given their respective inputs $\mathcal{I}$ and $\overline{\mathcal{I}}$. The corresponding prior $p(\mathbf{Z})$ is assumed as a joint of *i.i.d.* standard Gaussians, i.e.

$$p(\mathbf{Z}) = p(\mathbf{Z}^{\mathcal{I}})p(\mathbf{Z}^{\overline{\mathcal{I}}}) = \mathcal{N}(\mathbf{0},\mathbf{I})\mathcal{N}(\mathbf{0},\mathbf{I}). \tag{A.6}$$

So L_{recon} can now be considered as a negative of the ELBO bound derived as

$$\log\big(p_\theta(\mathbf{A},\mathbf{S})\big) = \log\Big(\int p(\mathbf{Z})p_\theta(\mathbf{A}|\mathbf{Z})p_\theta(\mathbf{S}|\mathbf{Z})d\mathbf{Z}\Big) \tag{A.7}$$

$$= \log\Big(\int \frac{p(\mathbf{Z})p_\theta(\mathbf{A}|\mathbf{Z})p_\theta(\mathbf{S}|\mathbf{Z})}{q_\phi(\mathbf{Z}|\mathcal{I},\overline{\mathcal{I}})}q_\phi(\mathbf{Z}|\mathcal{I},\overline{\mathcal{I}})d\mathbf{Z}\Big) \tag{A.8}$$

$$= \log\left(\mathbb{E}_{\mathbf{Z}\sim q}\Big\{\frac{p(\mathbf{Z})p_\theta(\mathbf{A}|\mathbf{Z})p_\theta(\mathbf{S}|\mathbf{Z})}{q_\phi(\mathbf{Z}|\mathcal{I},\overline{\mathcal{I}})}\Big\}\right) \tag{A.9}$$

$$\geq \mathbb{E}_{\mathbf{Z}\sim q}\left\{\log\Big(\frac{p(\mathbf{Z})p_\theta(\mathbf{A}|\mathbf{Z})p_\theta(\mathbf{S}|\mathbf{Z})}{q_\phi(\mathbf{Z}|\mathcal{I},\overline{\mathcal{I}})}\Big)\right\}, \tag{A.10}$$

where (A.10) follows from Jensen's Inequality. Using the factorizations in equation A.4 and equation A.5, we can now separate the factors inside log of (A.10) as

$$\mathbb{E}_{\mathbf{Z}\sim q}\left\{ \log\Big(\frac{p(\mathbf{Z})p_\theta(\mathbf{A}|\mathbf{Z})p_\theta(\mathbf{S}|\mathbf{Z})}{q_\phi(\mathbf{Z}|\mathcal{I},\overline{\mathcal{I}})}\Big)\right\}$$

$$= \mathbb{E}_{\mathbf{Z}\sim q}\left\{ \log\Big(\frac{p(\mathbf{Z}^{\mathcal{I}})p(\mathbf{Z}^{\overline{\mathcal{I}}})p_\theta(\mathbf{A}|\mathbf{Z})p_\theta(\mathbf{S}|\mathbf{Z})}{q_\phi(\mathbf{Z}^{\mathcal{I}}|\mathcal{I})q_\phi(\mathbf{Z}^{\overline{\mathcal{I}}}|\overline{\mathcal{I}})}\Big)\right\} \tag{A.11}$$

$$= \mathbb{E}_{\mathbf{Z}\sim q}\left\{ \log\Big(\frac{p(\mathbf{Z}^{\mathcal{I}})}{q_\phi(\mathbf{Z}^{\mathcal{I}}|\mathcal{I})}\Big) + \log\Big(\frac{p(\mathbf{Z}^{\overline{\mathcal{I}}})}{q_\phi(\mathbf{Z}^{\overline{\mathcal{I}}}|\overline{\mathcal{I}})}\Big) + \log\Big(p_\theta(\mathbf{A}|\mathbf{Z})\Big) + \log\Big(p_\theta(\mathbf{S}|\mathbf{Z})\Big)\right\} \tag{A.12}$$

$$= -D_{KL}\Big(q_\phi(\mathbf{Z}^{\mathcal{I}}|\mathcal{I})||p(\mathbf{Z}^{\mathcal{I}})\Big) - D_{KL}\Big(q_\phi(\mathbf{Z}^{\overline{\mathcal{I}}}|\overline{\mathcal{I}})||p(\mathbf{Z}^{\overline{\mathcal{I}}})\Big) - \mathrm{BCE}(\hat{\mathbf{A}},\mathbf{A}) - \mathrm{BCE}(\hat{\mathbf{S}},\mathbf{S}) \tag{A.13}$$

$$= -D_{KL}\Big(q_\phi(\mathbf{Z}^{\mathcal{I}}|\mathcal{I})||\mathcal{N}(\mathbf{0},\mathbf{I})\Big) - D_{KL}\Big(q_\phi(\mathbf{Z}^{\overline{\mathcal{I}}}|\overline{\mathcal{I}})||\mathcal{N}(\mathbf{0},\mathbf{I})\Big) - \mathrm{BCE}(\hat{\mathbf{A}},\mathbf{A}) - \mathrm{BCE}(\hat{\mathbf{S}},\mathbf{S}) \tag{A.14}$$

$$= \mathcal{L}_{\mathrm{ELBO}} = -L_{\mathrm{recon.}} \tag{A.15}$$

A.2 Detailed Comparison

We now aggregate the publicly available results for all three tasks discussed in the paper. The publicly available approaches often cover only a subset of the datasets evaluated in this work. So we leave the table cells empty in case of missing public results. In addition to the competitors mentioned in the paper, we add new competitors for different tasks.

A.2.1 Link Prediction

We add the following approaches in addition to the ones given in table 3.2: GNN based architectures **GCN** [40] and **GAT** [46], along with their hyperbolic variants **HGCN** [147] and **HGAT** [148]. Deep Generative Latent Feature Relational Model or **DGLFRM** [149] that aims to reconstruct the adjacency matrix while retaining the interpretability of stochastic block models. Graph InfoClust or **GIC** [104], which learns network embedding by maximizing the mutual information with respect to the graph-level summary as well as the cluster-level summaries. **Graphite** [150] is another autoencoder based generative model that employs a multi-layer procedure, inspired by low-rank approximations, to iteratively refine the reconstructed graph via message passing.

A.2.1.1 Results

We can notice that our approach is still either best or second best including all the competitors. GMI and GCA achieve good results for CoauthorCS, Graphite performs consistently well for all the datasets. However these algorithms suffer when evaluated for clustering and transductive node classification.

Algorithm	**Cora**		**CiteSeer**		**PubMed**		**WikiCS**		**CoauthorCS**		**AmazonComputers**		**AmazonPhoto**	
	AUC	**AP**	**AUC**	**AP**	**AUC**	**AP**	**AUC**	**AP**	**AUC**	**AP**	**AUC**	**AP**	**AUC**	**AP**
Spectral	84.60	88.50	80.50	85.00	84.20	87.80								
DeepWalk	83.10	85.00	80.50	83.60	84.40	84.10			91.74	91.19	87.35	91.48	91.48	90.79
GCN	90.47	91.62	82.56	83.20	89.56	90.28								
HGCN	92.90	93.45	95.25	95.97	96.30	96.75								
GAT	93.17	93.81	86.48	87.51	91.46	92.28								
HGAT	94.02	94.63	95.84	95.89	94.18	94.42								
GAE	91.00	92.00	89.50	89.90	96.40	96.50	93.00	94.80	94.09	93.86	93.61	93.36	93.86	92.96
VGAE	91.40	92.60	90.80	92.00	94.40	94.70	93.60	95.00	89.60	89.36	93.15	92.40	92.05	92.02
ARGA	92.40	93.20	91.90	93.00	96.80	97.10	93.40	94.70	91.99	92.54			**96.10**	**95.40**
ARVGA	92.40	92.60	92.40	93.00	96.50	96.80	94.70	94.80	93.32	93.32			92.70	90.90
DGLFRM	93.43	93.76	93.79	94.38	93.95	94.97								
GALA	92.10	92.20	94.40	94.80	91.50	89.70	93.60	93.10	93.81	94.49			91.80	91.00
Graphite	94.70	94.90	97.30	97.40	97.40	97.40								
DGI	89.80	89.70	95.50	95.70	91.20	92.20			94.87	94.34			92.24	92.14
GIC	93.50	93.30	97.00	96.80	93.70	93.50			95.03	94.94			92.70	92.34
GMI	95.10	95.60	97.80	97.40	96.37	96.04			96.37	95.04			93.88	92.67
GCA	95.75	95.47	96.44	96.49	95.28	95.52			96.31	**96.28**			93.25	92.74
MVGRL	90.52	90.45	92.89	92.89	92.45	92.17			95.17	95.58			92.89	92.45
G-BT	87.46	86.84	93.42	93.01	94.53	94.26	93.05	93.18	92.64	91.40	91.54	90.59	95.12	94.45
BGAE	99.02	98.82	98.59	**98.61**	97.78	97.68	97.23	97.69	96.31	95.44	95.01	94.04	95.01	94.24
BGAE + Att	**99.35**	**99.23**	98.56	98.57	**98.06**	**98.03**	97.73	98.03	**96.51**	95.65	**95.13**	**94.48**	95.18	94.49
BVGAE	97.87	97.62	**98.63**	98.57	97.93	97.89	97.62	97.42	96.12	95.13	94.66	94.01	94.61	94.29
BVGAE + Att	98.03	97.77	98.23	98.08	97.77	97.74	**97.85**	**98.05**	96.21	95.34	94.90	94.07	94.97	94.28

Table A.1: Link prediction performance, as evaluated by AUC and AP metrics. The best results are styled as bold and second best are underlined.

Algorithm	**Cora**	**CiteSeer**	**PubMed**	**WikiCS**	**CoauthorCS**	**CoauthorPhysics**	**AmazonComputers**	**AmazonPhoto**
K-means	32.10	30.50	0.10	18.20	64.20	48.90	16.60	28.20
Spectral	12.70	5.60	4.20					
DeepWalk	32.70	8.80	27.90					
BigClam	0.70	3.60	0.60					
DNGR	31.80	18.00	15.50					
RMSC	25.50	13.90	25.50					
TADW	44.10	29.10	0.10					
GAE	42.90	17.60	27.70	24.30	73.10	54.50	44.10	61.60
VGAE	43.60	15.60	22.90	26.10	73.30	56.30	42.30	53.00
ARGA	44.90	35.00	30.50	27.50	66.80	51.20	23.50	57.70
ARVGA	52.60	33.80	29.00	28.70	61.60	52.60	23.70	45.50
GALA	57.70	<u>44.10</u>	32.70					51.20
Graphite	54.12	42.42	32.40					
DGI	41.10	31.50	27.70	31.00	74.70	67.00	31.80	37.60
GIC	53.70	**45.30**	31.90					
GRACE	46.18	38.29	16.27	42.82	75.62	OOM	47.93	65.13
AGC	53.70	41.10	31.60					
GMI	50.33	38.14	26.20					
GMNN	53.72	41.73	31.77					
DAEGC	52.80	39.70	26.60					
DBGAN	56.00	40.70	32.40					48.50
GCA	55.70	37.40	28.90	29.90	73.50	59.40	42.60	34.40
MVGRL	60.90	44.00	31.50	26.30	74.00	59.40	24.40	34.40
BGRL				39.69	77.32	55.68	53.64	**68.41**
G-BT	43.40	41.57	29.52	27.46	74.37	59.80	65.55	52.39
BGAE	**62.42**	43.36	<u>38.46</u>	<u>45.80</u>	<u>80.10</u>	<u>68.01</u>	**66.98**	67.13
BGAE + Att	<u>62.27</u>	43.84	**38.59**	**46.93**	**80.30**	**68.12**	<u>66.93</u>	<u>67.43</u>
BVGAE	59.60	43.29	37.41	40.78	79.01	67.10	60.98	61.33
BVGAE + Att	59.82	43.27	37.47	40.86	79.42	67.06	61.44	61.62

Table A.2: Node clustering performance, as evaluated by NMI. The best results are styled as bold and second best are underlined. OOM refers to Out-of-Memory.

A.2.2 Clustering

We include the following clustering-specific competitors in addition to the ones given in table 3.4 and section A.2.1: **BigClam** [118] uses matrix factorization for community detection. **DNGR** [151] learns network embedding by using stacked denoising autoencoders. **RMSC** [152] introduces a multi-view spectral clustering approach to recover a low-rank transition probability matrix from the transition matrices corresponding to multiple views of input data. **TADW** [153] learns the network embedding by treating DeepWalk as matrix factorization and adding the features of vertices. **AGC** [154] performs attributed graph clustering by first obtaining smooth node feature representations via k-order graph convolution and then performing spectral clustering on the learned features. **DAEGC** [155] uses GAT to encode the importance of the neighboring nodes in the latent space such that both the reconstruction loss and the KL-divergence based clustering loss are minimized.

In addition, we include some well-known approaches for unsupervised network embedding. **DBGAN** [87] introduces a bidirectional adversarial learning framework to learn network embedding in such a way that the prior distribution is also estimated along with the adversarial learning. **GMI** [105] is an unsupervised approach to learn node representations while aiming to improve generalization performance via added contrastive regularization. **GMNN** [156] relies on a random field model, which can be trained with variational expectation maximization.

A.2.2.1 Results

Our approach performs the best overall, although GIC and BGRL achieve the best results for CiteSeer and AmazonPhoto respectively. Graphite performed well in table A.1, but for clustering, it is outperformed by many other competitors. The converse is true for GIC and GALA, which outperform BGAE for a single dataset in clustering, but fail to compete in the link prediction task. Similarly, DBGAN emerges as a decent competitor for node-clustering. However, as we will see in the next section, its performance degrades for the task of node classification. We cannot comment on BGRL because we could neither find its public implementation nor any publicly published results for link prediction on the selected datasets.

A.2.3 Transductive Node Classification

The competitors evaluated in table A.3 have already been introduced in the main paper and in section A.2.1 and section A.2.2. We exclude some methods that are specifically designed for clustering, because such methods perform poor on transductive node classification.

Algorithm	Cora	CiteSeer	PubMed	WikiCS	CoauthorCS	CoauthorPhysics	AmazonComputers	AmazonPhoto
Raw	47.87	49.33	69.11	71.98	90.37	93.58	73.81	78.53
DeepWalk	70.66	51.39	74.31	77.21	87.70	94.90	86.28	90.05
GAE	71.53	65.77	72.14	70.15	90.01	94.92	85.18	91.68
VGAE	75.24	69.05	75.29	75.63	92.11	94.52	86.44	92.24
ARGA	74.14	64.14	74.12	66.88	89.41	93.10	84.39	<u>92.68</u>
ARVGA	74.38	64.24	74.69	67.37	88.54	94.30	84.66	92.49
Graphite	82.10	71.00	79.30					
DGI	81.68	71.47	77.27	75.35	92.15	94.51	83.95	91.61
GIC	81.73	71.93	77.33	77.28	89.40	93.10	84.89	92.11
GRACE	80.04	71.68	79.53	**80.14**	92.51	94.70	87.46	92.15
AGC	70.90	71.89	68.91					
GMI	83.05	<u>73.03</u>	80.10	74.85	OOM	OOM	82.21	90.68
GMNN	82.78	71.54	80.60					
DBGAN	77.30	69.70	77.50					
GCA	82.10	71.30	80.20	78.23	92.95	**95.73**	88.94	92.53
MVGRL	82.90	72.60	79.40	77.52	92.11	92.11	87.52	91.74
BGRL	82.70	71.10	79.60	<u>79.98</u>	93.31	95.56	89.68	**92.87**
G-BT	80.80	73.00	80.00	76.65	92.95	95.07	88.14	92.63
BGAE	<u>83.51</u>	72.43	**81.84**	78.93	**93.76**	95.01	<u>92.24</u>	91.10
BGAE + Att	**83.60**	72.41	<u>80.95</u>	79.53	**93.76**	<u>95.64</u>	**92.44**	91.89
BVGAE	82.62	72.97	80.02	77.52	93.25	95.13	89.19	89.38
BVGAE + Att	82.57	**73.09**	80.25	77.82	93.15	95.60	89.91	89.98

Table A.3: Transductive node classification performance, as evaluated by accuracy. The best results are styled as bold and second best are underlined.

A.2.3.1 Results

GRACE, GMI, GALA, and BGRL perform well for WikiCS, CiteSeer, and AmazonPhoto. However our approach performs the best overall as we achieve the best or second-best results in 6 out of 8 datasets. This demonstrates the efficacy of our approach over a variety of tasks unlike many competitors that shine only in some of the target tasks.

A.3 Ablation Studies

We now observe how our approach is affected by changes in β from equation 3.1, λ from equation 3.9, and average-node degree used for sparsification of $\overline{\mathbf{S}}$ into $\mathbf{S}$. For sake of brevity, we only plot the results for transductive node classification task because link prediction and clustering follow a similar pattern. To emphasize the relative performance, the vertical axes correspond to the percentage accuracy scores relative to the ones reported in table 3.3.

A.3.1 Effect of β

To evaluate the effect of β in equation 3.1, we sweep β for the values across the set $\{0, 0.1, 1, 10, 100, 1000, 10000\}$.

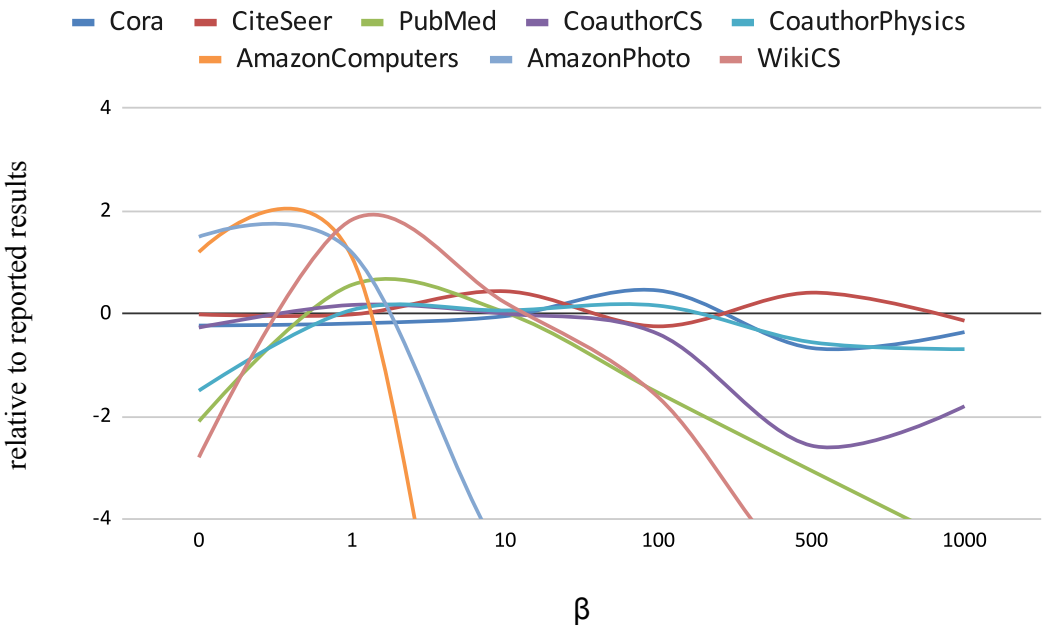


Figure A.1: Effect of β on transductive node classification performance. The vertical axis shows the performance in %, relative to the results reported in the main paper.

Figure A.3.1 shows the effect of β on transductive node classification for different datasets. For most of the datasets, the results are rather stable for quite a large range of β, i.e., in [1, 50] range. AmazonComputers and AmazonPhoto datasets are an exception in the sense that their performance degrades quicker than other datasets. Overall, a general trend of degradation can be observed for high values of β for all datasets, which is intuitive because for such values, the covariance loss takes over and the reconstruction loss is practically neglected, resulting in relatively poor results. Another observation is that the results above the 0-line on the graphs are better than the ones reported in the main paper. So, by carefully tuning β, we can achieve even better results compared to the ones reported in the paper.

A.3.2 Effect of λ

The hyperparameter λ governs the trade-off between invariance and cross-covariance in equation 3.9. The proposed value of λ in [53] is $5e^{-3}$. To see the effect of changing β, we sweep it across the values $\{1e^{-3}, 5e^{-3}, 1e^{-2}, 5e^{-2}\}$. The effect of changing λ on different datasets has been plotted in figure A.3.2. The plot validates that $\lambda = 5e^{-3}$, proposed in Barlow Twins [53], is a reasonable choice also for graph datasets. Some datasets perform better for $\lambda = 1e^{-3}$ and some yield better results for $\lambda = 1e^{-2}$. However, there is a general trend of decrease in the performance for $\lambda \geq 5e^{-2}$.

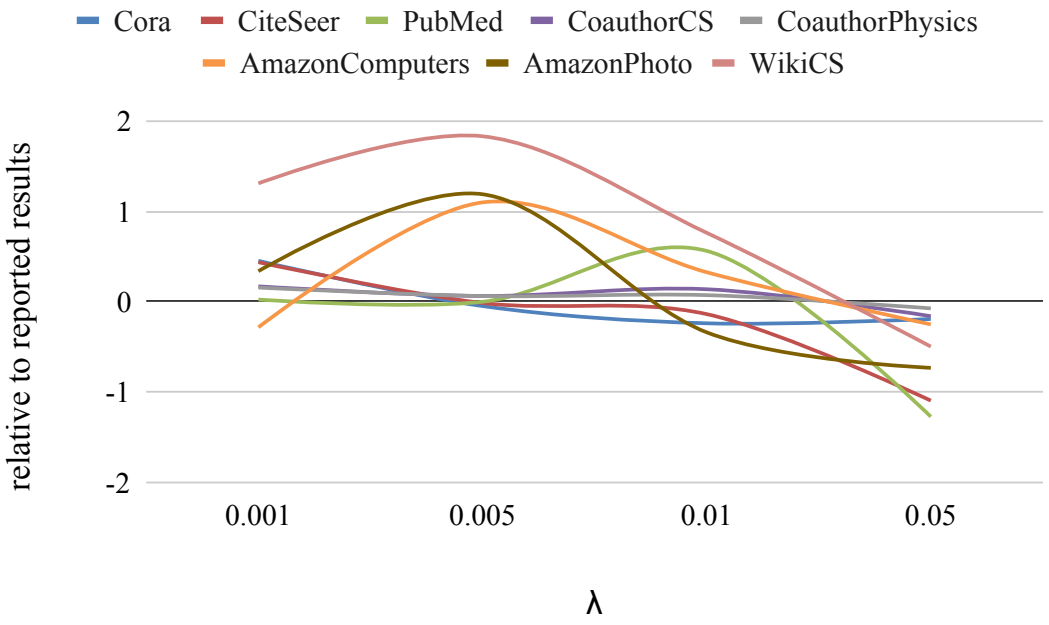


Figure A.2: Effect of λ on transductive node classification performance. The vertical axis shows the performance in %, relative to the results reported in the main paper.

A.3.3 Effect of Average Sparsification Degree in S

As mentioned in section 3.1.3, one of the ways of sparsification is to provide the intended average node degree. We have fixed this value to 25 for all the reported results. Now we observe the effect of changing this hyperparameter. We sweep the average degree over the values $\{5, 10, 20, 25, 30, 40, 50, 60, 75, 80, 100, 125, 150\}$. The results have been plotted in figure A.3.3. The results for PubMed are not plotted for the degree values greater than 75 because of out-of-memory issues. The relative performance remains more or less consistent over the plotted range, and varies between $\pm 1\%$ of the reported results. This also shows that the architecture can extract the relevant information from the neighborhood over a reasonable range of average degree. An exception is WikiCS where the results improve by up to 2% compared to the reported results in table 3.3 for the average degree value of 150. However, for such a high value, the graph is no longer reasonably sparse. This causes high training overhead because of the large number of edges in $\mathbf{S}$. On the other extreme, for the value of 5, we can see a decline in many datasets because here $\mathbf{S}$ is too sparse, hence the information in $\mathbf{S}$ is too little to be of use.

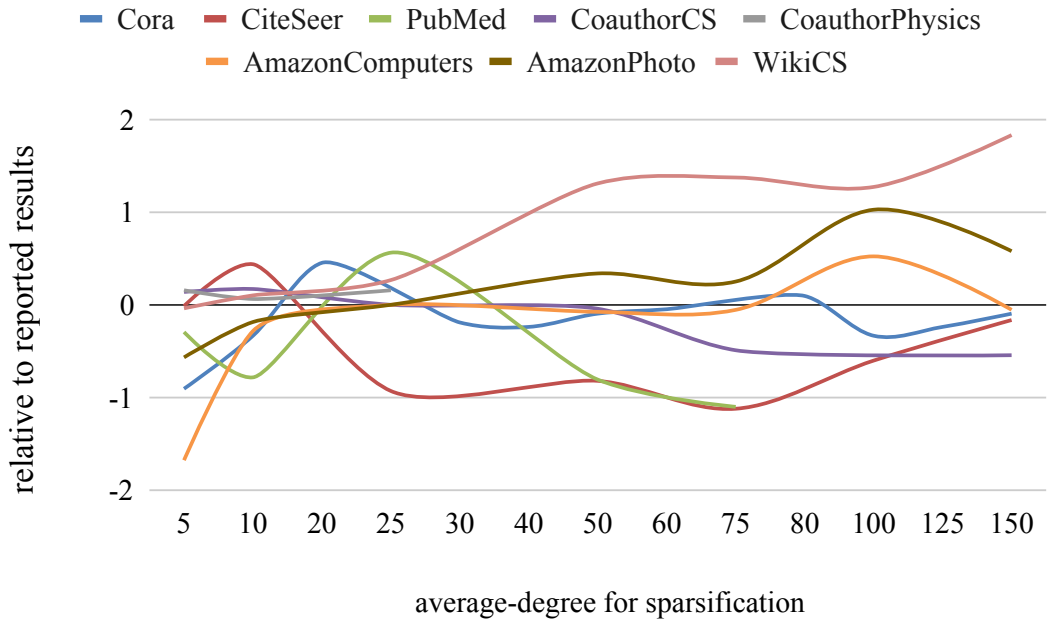


Figure A.3: Effect of average node degree in $\mathbf{S}$ on transductive node classification performance. The vertical axis shows the performance in %, relative to the results reported in the main paper.

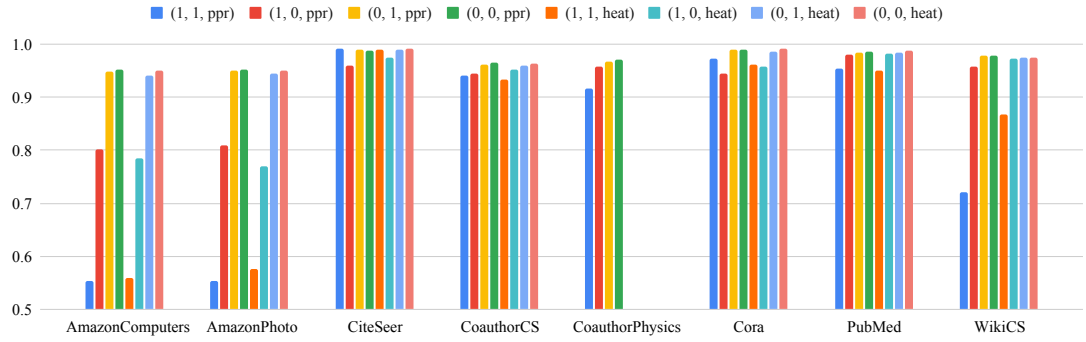
A.4 Variants of Our Approach

In the main paper, we have reported the results for PPR both with and without attention. From these results, we can already establish that it is always better to use attention i.e., let the neural network decide the weights for averaging the embeddings from the immediate and larger neighborhood. So, in this section, we focus on the case with attention, and report the results with following variations:

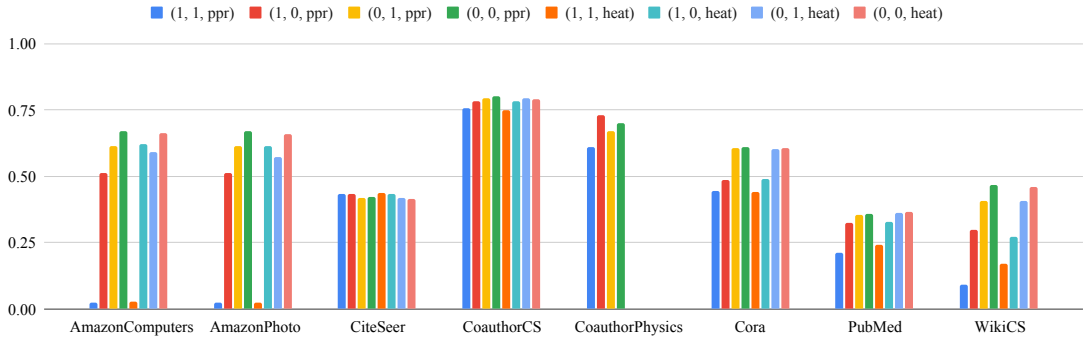
- Toggling L_{recon} on/off in equation 3.1
- Choosing between BGAE and BVGAE.
- Choosing between PPR and Heat Kernels for diffusion.

For brevity, we use triple of the form $\Big(\mathbf{1}(L_{\text{recon}} \text{ is mute}), \mathbf{1}(\text{ variational model}), \text{kernel name}\Big)$. For instance, (1, 0, ppr) means that we are referring to the variant where we are only using L_{cov} in non-variational mode with PPR kernel for diffusion. In the main paper, we have reported the results for the variant (0, 0, ppr) and (0, 1, ppr). Using this notation, we plot the results for all the eight variations for all three tasks i.e. link prediction, clustering and transductive node classification in figure A.4(a), figure A.4(b), and figure A.4(c) respectively. For some datasets (e.g., CoauthorPhysics), some variants could not be plotted because of out-of-memory issues. The general behavior is similar for different variants across all three tasks. The important observations from figure A.4 are as follows:

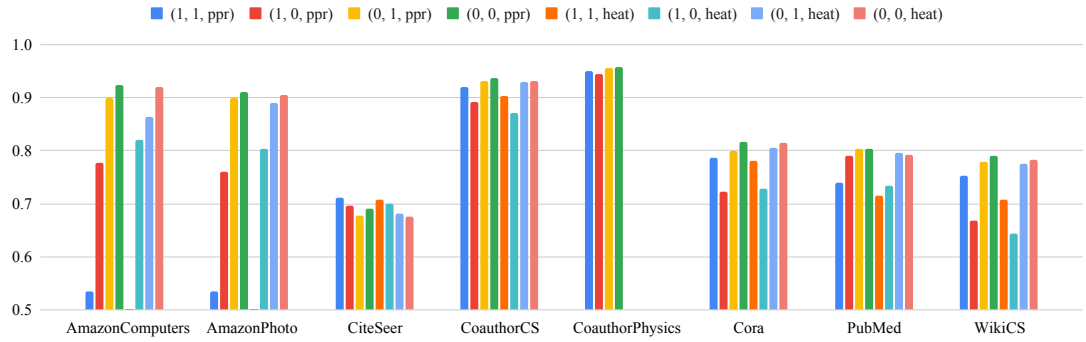
- The variant (0, 0, ppr), shown in green, performs the best overall.
- The variants (0, 0, ppr) and (0, 0, heat) are usually close in performance, although (0, 0, ppr) is often better by a small margin.
- The variants (0, 0, ppr) and (0, 0, heat) with simple GCN encoders usually outperform their variational counterparts, i.e., (0, 1, ppr) and (0, 1, heat). There are, however, minor exceptions. For instance, in figure A.4(a), (0, 1, ppr) is marginally better than (0, 0, ppr) for CiteSeer. Similarly, in figure A.4(b), (0, 1, heat) is marginally better than (0, 0, heat) for CoauthorCS.
- When L_{recon} is turned off, the performance is usually relatively worse than when L_{recon} is on. This can be seen in (1, 1, ppr), (1, 0, ppr), (1, 1, heat), and (1, 0, heat) variants. The only exception is CiteSeer in figure A.4(a) where (1, 1, ppr) outperforms (0, 1, ppr) by a tiny margin. This validates our intuition that L_{recon} aids L_{cov} almost always.



(a) Variants of our approach for link prediction



(b) Variants of our approach for node clustering



(c) Variants of our approach for node classification

Figure A.4: Variants of BGAE for all link prediction, node clustering, and node classification.

B ELBO Bound for J-ENC and VaCA-HINE

$$
\begin{aligned}
&\log(p_\theta(\boldsymbol{A})) \\
&= \log\Big(\int \sum_{\boldsymbol{c}} p_\theta(\boldsymbol{Z}, \boldsymbol{c}, \boldsymbol{A}) d\boldsymbol{Z} \Big) && \text{(B.1)} \\
&= \log\Big(\int \sum_{\boldsymbol{c}} p(\boldsymbol{Z})\ p_\theta(\boldsymbol{c}|\boldsymbol{Z})\ p_\theta(\boldsymbol{A}|\boldsymbol{c}, \boldsymbol{Z}) d\boldsymbol{Z} \Big) && \text{(B.2)} \\
&= \log\Big(\mathbb{E}_{(\boldsymbol{Z},\boldsymbol{c})\sim q_\phi(\boldsymbol{Z},\boldsymbol{c}|\mathcal{I})} \Big\{ \frac{p(\boldsymbol{Z})\ p_\theta(\boldsymbol{c}|\boldsymbol{Z})\ p_\theta(\boldsymbol{A}|\boldsymbol{c}, \boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{I}) q_\phi(\boldsymbol{c}|\boldsymbol{Z}, \mathcal{I})} \Big\} \Big) && \text{(B.3)} \\
&\geq \mathbb{E}_{(\boldsymbol{Z},\boldsymbol{c})\sim q_\phi(\boldsymbol{Z},\boldsymbol{c}|\mathcal{I})} \Big\{ \log\Big(\frac{p(\boldsymbol{Z})\ p_\theta(\boldsymbol{c}|\boldsymbol{Z})\ p_\theta(\boldsymbol{A}|\boldsymbol{c}, \boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{I}) q_\phi(\boldsymbol{c}|\boldsymbol{Z}, \mathcal{I})} \Big) \Big\} && \text{(B.4)} \\
&= \mathbb{E}_{(\boldsymbol{Z},\boldsymbol{c})\sim q_\phi(\boldsymbol{Z},\boldsymbol{c}|\mathcal{I})} \Big\{ \log\Big(\frac{p(\boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{I})} \Big) + \log\Big(\frac{p_\theta(\boldsymbol{c}|\boldsymbol{Z})}{q_\phi(\boldsymbol{c}|\boldsymbol{Z}, \mathcal{I})} \Big) + \log\Big(p_\theta(\boldsymbol{A}|\boldsymbol{c}, \boldsymbol{Z}) \Big) \Big\} && \text{(B.5)} \\
&= \mathbb{E}_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\mathcal{I})} \Big\{ \log\Big(\frac{p(\boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{I})} \Big) \Big\} \\
&\quad + \mathbb{E}_{(\boldsymbol{Z},\boldsymbol{c})\sim q_\phi(\boldsymbol{Z},\boldsymbol{c}|\mathcal{I})} \Big\{ \log\Big(\frac{p_\theta(\boldsymbol{c}|\boldsymbol{Z})}{q_\phi(\boldsymbol{c}|\boldsymbol{Z}, \mathcal{I})} \Big) \Big\} \\
&\quad + \mathbb{E}_{(\boldsymbol{Z},\boldsymbol{c})\sim q_\phi(\boldsymbol{Z},\boldsymbol{c}|\mathcal{I})} \Big\{ \log\Big(p_\theta(\boldsymbol{A}|\boldsymbol{c}, \boldsymbol{Z}) \Big) \Big\}. && \text{(B.6)}
\end{aligned}
$$

Where (B.4) follows from Jensen's Inequality. First term of (B.6) is given by:

$$
\begin{aligned}
\mathbb{E}_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\mathcal{I})} \Big\{ \log\Big(\frac{p(\boldsymbol{Z})}{q_\phi(\boldsymbol{Z}|\mathcal{I})} \Big) \Big\} &= \sum_{i=1}^{N} \mathbb{E}_{\mathbf{z}_i\sim q_\phi(\mathbf{z}_i|\mathcal{I})} \Big\{ \log\Big(\frac{p(\mathbf{z}_i)}{q_\phi(\mathbf{z}_i|\mathcal{I})} \Big) \Big\} && \text{(B.7)} \\
&= -\sum_{i=1}^{N} D_{KL}(q_\phi(\mathbf{z}_i|\mathcal{I}) \ ||\ p(\mathbf{z}_i)). && \text{(B.8)}
\end{aligned}
$$

Second term of equation B.6 can be derived as:

$$\begin{aligned}
&\mathbb{E}_{(\boldsymbol{Z},\boldsymbol{c})\sim q_\phi(\boldsymbol{Z},\boldsymbol{c}|\mathcal{I})}\left\{\log\left(\frac{p_\theta(\boldsymbol{c}|\boldsymbol{Z})}{q_\phi(\boldsymbol{c}|\boldsymbol{Z},\mathcal{I})}\right)\right\} \\
&= \sum_{i=1}^{N}\mathbb{E}_{(\mathbf{z}_i,c_i)\sim q_\phi(\mathbf{z}_i,c_i|\mathcal{I})}\left\{\log\left(\frac{p_\theta(c_i|\mathbf{z}_i)}{q_\phi(c_i|\mathbf{z}_i,\mathcal{I})}\right)\right\} && \text{(B.9)} \\
&\approx \sum_{i=1}^{N}\frac{1}{\mathrm{J}}\sum_{j=1}^{\mathrm{J}}\mathbb{E}_{c_i\sim q_\phi(c_i|\mathbf{z}_i^{(j)},\mathcal{I})}\left\{\log\left(\frac{p_\theta(c_i|\mathbf{z}_i^{(j)})}{q_\phi(c_i|\mathbf{z}_i^{(j)},\mathcal{I})}\right)\right\} && \text{(B.10)} \\
&= -\sum_{i=1}^{N}\frac{1}{\mathrm{J}}\sum_{j=1}^{\mathrm{J}}D_{KL}(q_\phi(c_i|\mathbf{z}_i^{(j)},\mathcal{I})\;||\;p_\theta(c_i|\mathbf{z}_i^{(j)})) && \text{(B.11)}
\end{aligned}$$

where (B.10) follows from equation B.9 by replacing the expectation over $\mathbf{z}_i$ with sample mean by generating J samples $\mathbf{z}_i^{(j)}$ from distribution $q(\mathbf{z}_i|\mathcal{I})$. Assuming $\mathrm{a}_{ij} \in [0,1]\ \forall i$, the third term of equation B.6 is the negative of binary cross entropy (BCE) between observed and predicted edges.

$$\begin{aligned}
&\mathbb{E}_{(\boldsymbol{Z},\boldsymbol{c})\sim q_\phi(\boldsymbol{Z},\boldsymbol{c}|\mathcal{I})}\left\{\log\Big(p_\theta(\boldsymbol{A}|\boldsymbol{c},\boldsymbol{Z})\Big)\right\} \\
&\quad= \sum_{(i,j)\in\mathcal{E}}\mathbb{E}_{(\mathbf{z}_i,\mathbf{z}_j,c_i,c_j)\sim q_\phi(\mathbf{z}_i,\mathbf{z}_j,c_i,c_j|\mathcal{I})}\left\{\log\Big(p_\theta(\mathrm{a}_{ij}|c_i,c_j,\mathbf{z}_i,\mathbf{z}_j)\Big)\right\} && \text{(B.12)}
\end{aligned}$$

Hence, by substituting equation B.8 and equation B.11 in equation B.6, we get the ELBO bound as:

$$\begin{aligned}
\mathcal{L}_{ELBO} \approx &-\sum_{i=1}^{N}D_{KL}(q_\phi(\mathbf{z}_i|\mathcal{I})\;||\;p(\mathbf{z}_i)) \\
&-\sum_{i=1}^{N}\frac{1}{\mathrm{J}}\sum_{j=1}^{\mathrm{J}}D_{KL}(q_\phi(c_i|\mathbf{z}_i^{(j)},\mathcal{I})\;||\;p_\theta(c_i|\mathbf{z}_i^{(j)})) \\
&+\sum_{(i,j)\in\mathcal{E}}\mathbb{E}_{(\mathbf{z}_i,\mathbf{z}_j,c_i,c_j)\sim q_\phi(\mathbf{z}_i,\mathbf{z}_j,c_i,c_j|\mathcal{I})}\left\{\log\Big(p_\theta(\mathrm{a}_{ij}|c_i,c_j,\mathbf{z}_i,\mathbf{z}_j)\Big)\right\} \qquad \text{(B.13)}
\end{aligned}$$

C VaCA-HINE Supplementary Material

C.1 Implementation Details of VaCA-HINE

The training process of VaCA-HINE has the following steps:

1. **Z Initialization:** This step involves pre-training of the variational encoder to get $\mathbf{Z}$. So, L_{enc} is the only loss considered in this step.

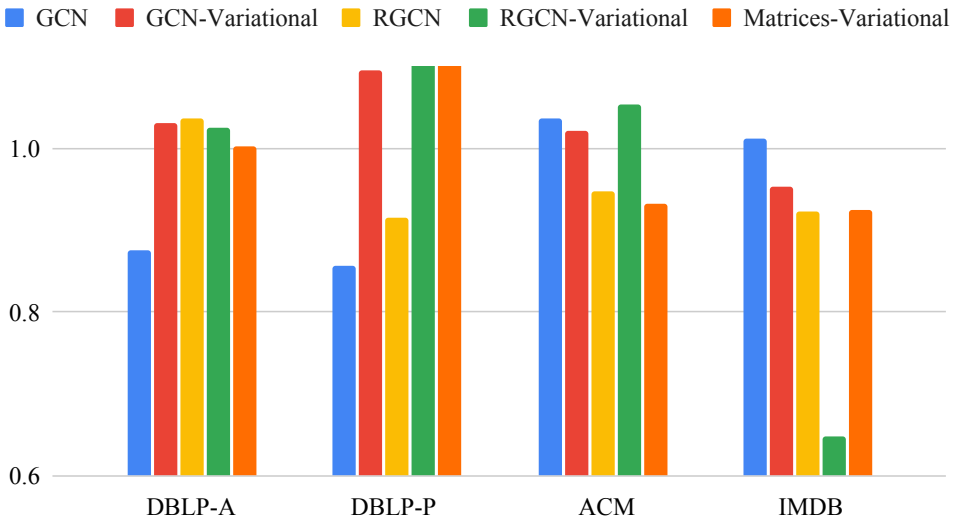
2. $\mathbf{g}_k$ **Initialization:** We fit K-Means on $\mathbf{Z}$ and then transform $\mathbf{Z}$ to the community-distance space, i.e., we get a transformed matrix of size $N \times K$ where the entry in i-th row and k-th column is the euclidean distance of $\mathbf{z}_i$ from k-th community center. A softmax over the columns of this matrix gives us initial probabilistic community assignments. We then pre-train the community embeddings $\mathbf{g}_k$ by minimizing the KL-divergence between $p_\theta(\mathbf{c}|\mathbf{Z})$ and the initialized community assignment probabilities. This KL-divergence term is used as a substitute for L_{c}. All the other loss terms function the same as detailed in section 5.4 and section 5.5

3. **Joint Training:** The end-to-end training of VaCA-HINE is performed in this step to minimize the loss in equation 5.1.

C.2 VaCA-HINE with Different Encoders

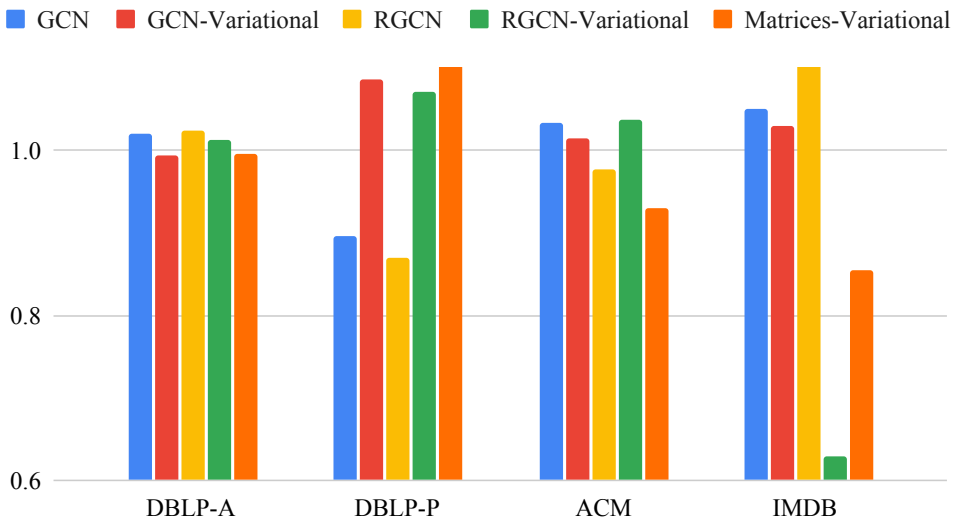
In this section, we evaluate the performance of VaCA-HINE for five different encoders. For GCN [40] and RGCN [62], we compare both variational and non-variational counterparts. For variational encoder, we learn both mean and variance, followed by the reparameterization trick as given in section 5.4.2.1. For non-variational encoders, we ignore L_{enc} as we directly learn $\mathbf{Z}$ from the input $\mathcal{I}$. In addition to GCN and RGCN encoders, we also give the results for a simple linear encoder consisting of two matrices to learn the parameters of $q_\phi(\mathbf{z}_i|\mathcal{I})$. For classification, we only report the F1-Micro score as F1-Macro follows the same trend.

The relative community detection and classification performances achieved using these encoders are illustrated in figure C.1 and figure C.2 respectively. The results in these bar plots are normalized by the second best values in table 5.2 and table 5.3 for better visualization. In figure C.1 the results are usually better with L_{cont}, although the performance difference is rather small. One exception is the IMDB



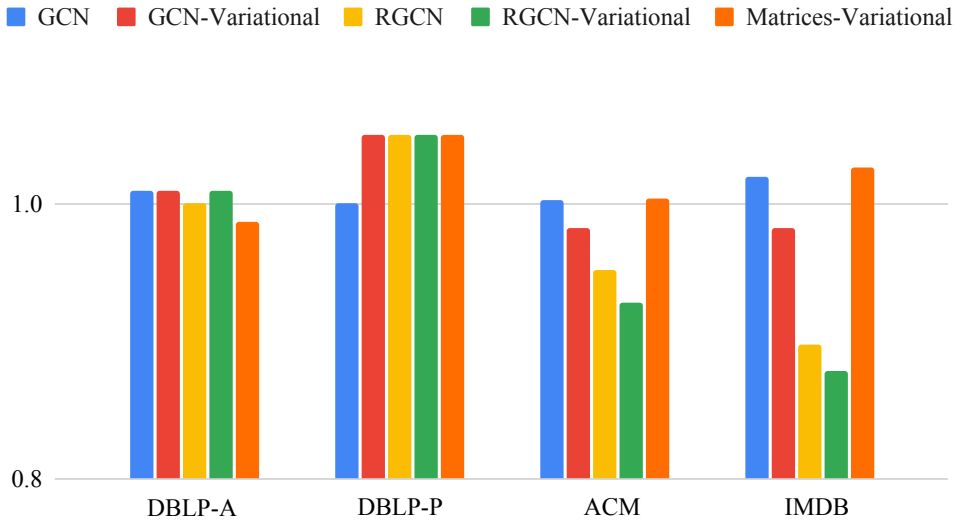
chapter

(a) With L_{cont}.

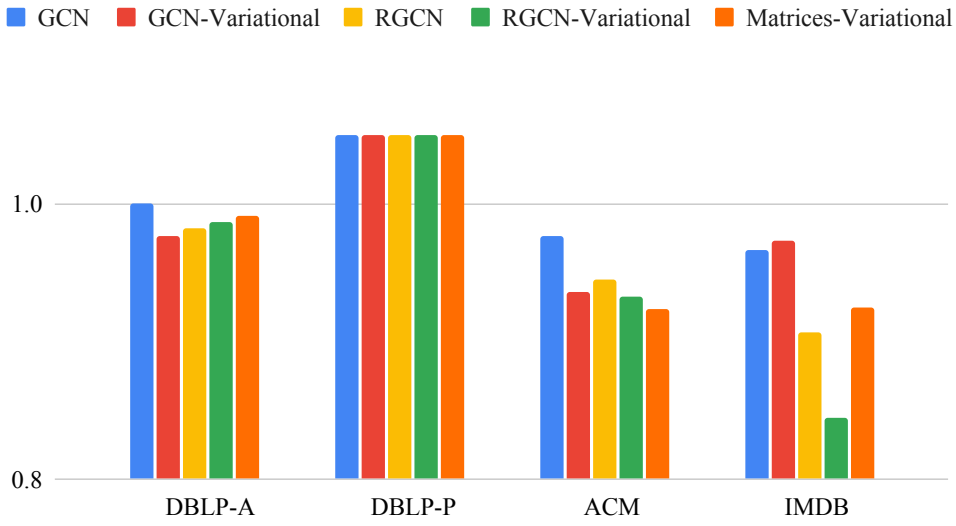


(b) Without L_{cont}.

Figure C.1: Community Detection performance (NMI) of VACA-HINE for different types of encoders.



(a) With L_{cont}.



(b) Without L_{cont}.

Figure C.2: Classification performance (F1-Micro) of VACA-HINE for different types of encoders.

dataset where the results are better without L_{cont} as stated in section 5.6.4. Overall, VaCA-HINE performs better than its competitors with at least three out of five encoders in all four cases. The effect of L_{cont} gets more highlighted for the classification task in figure C.2. For instance, without L_{cont}, VaCA-HINE fails to beat the second-best F1-Micro scores for ACM and IMDB, irrespective of the chosen encoder architecture. In addition, it is worth noticing that even a simple matrices-based variational encoder yields reasonable performance for downstream classification, which hints at the stability of the architecture of VaCA-HINE for downstream node classification even with a simple encoder.