A Variational Edge Partition Model for Supervised Graph Representation Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph neural networks (GNNs), which propagate the node features through the edges and learn how to transform the aggregated features under label supervision, have achieved great success in supervised feature extraction for both node-level and graph-level classification tasks. However, GNNs typically treat the graph structure as given and ignore how the edges are formed. This paper introduces a graph generative process to model how the observed edges are generated by aggregating the node interactions over a set of overlapping node communities, each of which contributes to the edges via a logical OR mechanism. Based on this generative model, we partition each edge into the summation of multiple community-specific weighted edges and use them to define community-specific GNNs. A variational inference framework is proposed to jointly learn a GNN-based inference network that partitions the edges into different communities, these community-specific GNNs, and a GNN-based predictor that combines community-specific GNNs for the end classification task. Extensive evaluations on real-world graph datasets have verified the effectiveness of the proposed method in learning discriminative representations for both node-level and graph-level classification tasks.

1 Introduction

Many real-world entities are bonded by relations, *e.g.*, the users in a social network service are connected by online friendships and the atoms in molecules are held together by chemical bonds. Representing the entities by nodes and relations by edges, a set of interconnected entities naturally forms a graph. Reasoning about these entities and their relations could be conducted under graph neural networks (GNNs) [1–4]. Instead of isolating the feature transformation for each individual entity, GNNs allow the information to be exchanged between related entities. Models employing this strategy have achieved great success in a wide range of applications involving graph structured data, such as classification [5, 6], link prediction [7], recommendation [8, 9], (overlapping) community detection [10, 11], and drug discovery [12].

The essence of many graph-analytic tasks could be summarized as supervised graph representation learning, where the information on the graph is usually characterized by the node features, adjacency matrix, and node or graph labels. A supervised machine learning pipeline is often introduced to embed the nodes or graph under label supervision for a specific classification task. For most of the GNN-based methods, the information on nodes and edges are composited through neighborhood aggregation, *i.e.*, updating the features of each node by combining the features of the surrounding nodes. During this process, the graph structure mostly serves as a given source to indicate the nodes' neighborship [13] or provide the weights for aggregation [5]. Treating the graph structure as given overlooks the latent structures that control the formation of the graph, which, however, could provide

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

valuable information for graph representation learning. In other words, failing to consider the latent graph structures may limit the ultimate potential of this line of work.

One type of latent structure at the hinge of the graph structure and node information is node communities. Their connection to the graph can be explained under a latent-community based graph generation process. For example, both the mixed-membership stochastic blockmodel (MMSB) [14] and edge partition model (EPM) [15] explain the formation of edges by node interactions over overlapping latent communities. Let us consider person u in social network $\mathcal{G}$, some of her social connections may be established through interactions with colleagues as a machine learning researcher, some may originate from her interactions with co-members of the same hobby club, and different types of interactions could overlap to strengthen certain social connections between the nodes.

The node community structure also sets the aspects of node information. In the example of social network $\mathcal{G}$ where person u is affiliated with both a research group and a few hobby groups, due to the diverse nature of these different communities, it is likely that she exhibits different characteristics when interacting with co-members of different communities. Namely, the information on u for the research group may be related to her research expertise, and the information on u for the hobby clubs may be related to her hobbies. Therefore, an ideal solution is to learn a different aspect of node properties for each community that it is affiliated with, and represent the overall node information as an aggregation of all community-specific node properties.

Based on this insight, we develop a *variational edge partition model* (VEPM), which is a generative graph representation learning framework. VEPM models how the edges and labels are generated from overlapping latent communities. Instead of hard assigning a node to a single community, VEPM encodes each node into a K-dimensional vector of scores measuring the strength that the node is affiliated with each of the K communities. From these scores we compute the intensities of pairwise interactions within each community. Given that information, under the Bernoulli-Poisson link, the edges could be modeled by the logical OR of independently generated binary latent edges [15].

The generation of labels includes community-specific node representation learning and aggregation. A key premise of the first part is to detect the hidden structure specified for each community. In our model, it is achieved through edge partition, *i.e.*, decomposing each edge into a summation of link strengths according to the intensities of community-specific interactions. The edge partition step isolates the link strengths accumulated through each community. With the partitioned edges, we learn K separate GNNs to produce community-specific node representations, then compose them into the overall node embeddings to generate the labels. Evaluation shows that the proposed framework can achieve significant performance enhancement on various node and graph classification benchmarks. We summarize our main contributions as follows:

- We introduce VEPM that utilizes the idea of edge partitioning to extract overlapping latent community structures, which are used to not only enrich node attributes with node-community affiliation scores, but also define community-specific node feature aggregations.
- We formalize the training of VEPM in a variational inference framework, which is powered by GNNs in its latent community inference, representation generation, and label prediction. We further develop an effective pretrain-finetune algorithm for training VEPM.
- We analyze how VEPM works and evaluate it over various real-world network datasets. Empirical results show that the proposed VEPM outperforms many previous methods in various node- and graph-level classification tasks, especially with limited labels.

2 Variational Edge Partition Model

The information of a node-attributed graph $\mathcal{G}$ with N nodes could generally be summarized by the duad $(\mathbf{X}, \mathbf{A})$, where $\mathbf{X} \in \mathbb{R}^{N \times F}$ is the design matrix whose rows represent node features of dimension F, and $\mathbf{A}$ is an adjacency matrix of shape $N \times N$. The indices of nonzero entries of rows in $\mathbf{A}$ indicate the neighbors of the corresponding nodes, and the values are the initial (unnormalized) edge weights. Structured data numerically expressed in this way could be processed by GNNs, which typically are deep architectures consisting of multiple GNN layers. Denoting the embedding of node u at the exit of hidden layer l as $\mathbf{h}_u^{(l)}$, the propagation rule of a GNN layer could be summarized as

$$\mathbf{h}_u^{(l)} = \mathrm{agg}\Big(f_{\boldsymbol{\theta}}(\mathbf{h}_u^{(l-1)}), \big\{f_{\boldsymbol{\theta}}(\mathbf{h}_v^{(l-1)})\big\}_{v \in \mathbb{N}(u)}\Big), \tag{1}$$

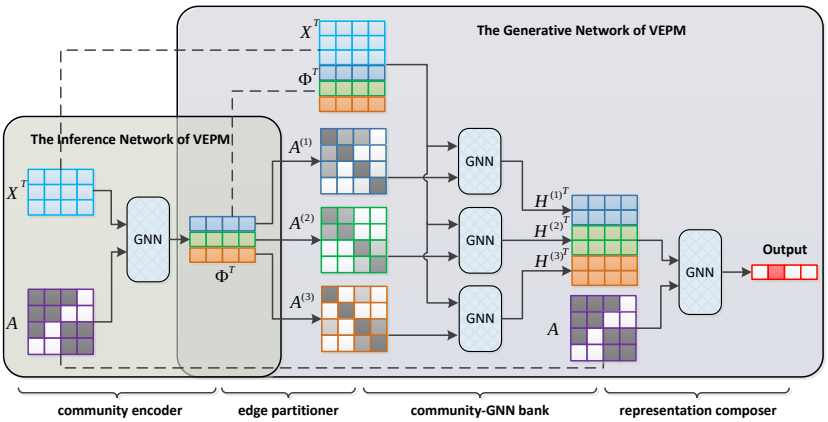


Figure 1: The overview of VEPM's computation graph. $\mathbf{\Phi}$ encodes nodes' affiliation strength to each community, which is used to (1) enrich node features with community information (the dashed line) and (2) partition edges (the solid arrows). The weighted edges obtained from graph partitioning are then passed to an array of independent GNNs to learn community-specific node representations. Finally, they are ensembled to estimate the predictive distribution of the object categories.

where $\mathrm{agg}(\cdot)$ is the neighborhood aggregation function, $\mathbb{N}(u)$ denotes the neighbors of u, and $f_{\boldsymbol{\theta}}(\cdot)$ is a transformation function parameterized with $\boldsymbol{\theta}$. Models consisting of GNN layers as described in Equation (1) are collectively referred to as $\mathrm{GNN}_{\boldsymbol{\theta}}(\mathbf{X}, \mathbf{A})$.

Many analytical tasks on node-attributed graphs could boil down to (semi-) supervised classification problems, *i.e.*, given $(\mathbf{X}, \mathbf{A})$ and observed node or graph labels $\mathbf{y}_o$, predicting the unobserved labels $\mathbf{y}_u$. Note while the unlabeled objects are not limited to node-level, the prediction task typically becomes tied to infer node-level representations. In VEPM, we formalize the classification task as modeling the predictive distribution as $p(\mathbf{y}_u \,|\, \mathbf{X}, \mathbf{A}, \mathbf{y}_o)$, and construct our variational supervised learning pipeline with GNNs.

2.1 Generative task-learning with latent communities

To model $p(\mathbf{y}_u \,|\, \mathbf{X}, \mathbf{A}, \mathbf{y}_o)$, we introduce $\mathbf{\Phi} \in \mathbb{R}_+^{N \times K}$, a latent non-negative node-community affiliation matrix, whose entry at the nth row and kth column is interpreted as the strength that the nth node is affiliated with the kth community. Given $\mathbf{\Phi}$, we specify a *generative network* as

$$\mathbf{A} \sim p_{\boldsymbol{\theta}}(\mathbf{A} \,|\, \mathbf{\Phi}), \quad \mathbf{y}_o \sim p_{\boldsymbol{\theta}}(\mathbf{y} \,|\, \mathbf{X}, \mathbf{A}, \mathbf{\Phi}) \tag{2}$$

to describe how the observed edges $\mathbf{A}$ and labels $\mathbf{y}_o$ are generated. Due to the complexity brought by the deep architecture of the *generative network*, analytically inferring the posterior of $\mathbf{\Phi}$ is impractical, we hence approximate the posterior $p_{\boldsymbol{\theta}}(\mathbf{\Phi} \,|\, \mathbf{A}, \mathbf{y}_o)$ with a variational distribution, $q_{\boldsymbol{\omega}}(\mathbf{\Phi} \,|\, \mathbf{A}, \mathbf{X})$, modeled by a separate GNN-based *inference network*. The subscripts $_{\boldsymbol{\theta}}$ and $_{\boldsymbol{\omega}}$ denote the parameters in the *generative network* and *inference network*, respectively. We illustrate the architecture of VEPM in Figure 1. We approximate the posterior predictive distribution using the Monte Carlo method as

$$\hat{p}_{\boldsymbol{\theta}}(\mathbf{y}_u \,|\, \mathbf{X}, \mathbf{A}, \mathbf{y}_o) \approx \tfrac{1}{S} \textstyle\sum_{s=1}^{S} p_{\boldsymbol{\theta}}(\mathbf{y}_u \,|\, \mathbf{X}, \mathbf{A}, \mathbf{\Phi}^{(s)}), \tag{3}$$

where $\mathbf{\Phi}^{(s)} \overset{iid}{\sim} q_{\boldsymbol{\omega}}(\mathbf{\Phi} \,|\, \mathbf{A}, \mathbf{X})$ for $s = 1, \ldots, S$.

In what follows, we outline the data generation process and corresponding modules in the *generative network*, introduce the module in the *inference network*, and describe how to train both networks.

2.2 The generation of edges and labels

To model the distribution of the edges given $\mathbf{\Phi}$, we adopt the generative process developed in EPM [15], which explains the generation of the edges under the Bernoulli-Poisson link as $\mathrm{A}_{i,j} = \mathbf{1}_{\mathrm{M}_{i,j} \geq 1}$, $\mathrm{M}_{i,j} \sim \mathrm{Poisson}\left(\sum_{k=1}^{K} \gamma_k \Phi_{i,k} \Phi_{j,k}\right)$. Here γ_k is a positive community activation level indicator, which measures the member interaction frequency via community k, and in our practice is

treated as a trainable parameter shared by both the *inference network* and *generative network*. We interpret $\gamma_k \Phi_{i,k} \Phi_{j,k}$ as the interaction rate between nodes i and j via community k.

The EPM has an alternative representation that partitions each edge into the logical disjuction ($i.e.$, logical OR) of K latent binary edges, expressed as

$$\mathrm{A}_{i,j} = \vee_{k=1}^{K} \mathrm{A}_{i,j,k}, \ \mathrm{A}_{i,j,k} \sim \mathrm{Ber}(p_{ijk}),$$

where $p_{ijk} := 1 - e^{-\gamma_k \Phi_{i,k} \Phi_{j,k}}$. Thus nodes i and j would be connected as long as their interaction forms up an edge in at least one community. In other words, they are disconnected only if their interactions in all communities fail to generate any edges. Under the EPM, we have the conditional probability as $p_{\boldsymbol{\theta}}(\mathrm{A}_{i,j} = 1 \,|\, \boldsymbol{\Phi}) = 1 - e^{-\sum_{k=1}^{K} \gamma_k \Phi_{i,k} \Phi_{j,k}}$. To complete the model, we set the prior distribution of the node-community affiliation matrix as $\boldsymbol{\Phi} \sim \prod_{i=1}^{N} \prod_{k=1}^{K} \mathrm{Gam}(\Phi_{i,k}; \alpha, \beta)$.

Given $\mathbf{A}$ and $\boldsymbol{\Phi}$, we now describe the generation process of the labels. The node-community affiliation impacts this process from two aspects: for each node, the corresponding row of $\boldsymbol{\Phi}$ serves as side information that could enrich node attributes; for each pair of connected nodes, it could be used to derive the node interactions rate via each community, *i.e.*, $\{\gamma_k \Phi_{i,k} \Phi_{j,k}\}_{k=1,K}$, which are further used to partition the edges to K communities, carried out by the *edge partitioner*:

Edge partitioner. *The edge partitioner takes edges $\mathbf{A}$ and node-community affiliation matrix $\boldsymbol{\Phi}$ as inputs and returns K positive-weighted edges: $\{\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(K)}\}$, s.t. $\sum_{k=1}^{K} \mathbf{A}^{(k)} = \mathbf{A}$. The edge partition function is*

$$\mathrm{A}_{i,j}^{(k)} = \mathrm{A}_{i,j} \cdot \frac{e^{(\gamma_k \Phi_{i,k} \Phi_{j,k})/\tau}}{\sum_{k'} e^{(\gamma_{k'} \Phi_{i,k'} \Phi_{j,k'})/\tau}}, \ k \in \{1, \ldots, K\}, \ i, j \in \{1, \ldots, N\}, \tag{4}$$

where τ is a "temperature" that controls the sharpness of partition.

The values of τ are specified in Appendix C. The effect of the *edge partitioner* could be considered as a soft assignment of each edge into different communities. Setting temperature τ to be low could drive the soft assignment towards a hard assignment, and consequently, when aggregating node features with $\{\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(K)}\}$, the edge partitioner would concentrate the information exchange between any two connected nodes at one community. In other words, the mutual influence between two nodes, measured by the edge weight, is relatively high at the community that contributes the most of the interactions between the pair, while being relatively weak at the other communities.

In our implementation, we further generalize Equation (4) to a metacommunity-based edge partition, specifically, by replacing $\gamma_k \Phi_{i,k} \Phi_{j,k}$ with $\boldsymbol{\Phi}_{i,k} \mathrm{diag}(\boldsymbol{\gamma}_k) \boldsymbol{\Phi}'_{j,k}$. In the new expression, $\boldsymbol{\Phi}_{i,k}$ denotes the kth segment in node i's community-affiliation encoding, $\boldsymbol{\gamma}_k$ is a vector of the activation levels for communities in the kth metacommunity. This generalization allows the total number of communities to be greater than K, enhancing the model implementation flexibility at a minor interpretability cost.

The *edge partitioner* provides a soft separation of the information propagation routes for each community, which is passed to the *community-GNN bank* described below for community-specific node representation learning:

Community-GNN bank. *The module takes $\mathbf{X}^* := \mathbf{X} \,\|\, \boldsymbol{\Phi}$ as input, where the operator $\|$ denotes "concatenation," and learns community-specific node embeddings with K separate GNNs, i.e., the outputs are $\mathrm{g}_{\boldsymbol{\theta}}^{(1)}(\mathbf{X}^*), \ \mathrm{g}_{\boldsymbol{\theta}}^{(2)}(\mathbf{X}^*), \ \cdots, \ \mathrm{g}_{\boldsymbol{\theta}}^{(K)}(\mathbf{X}^*)$, where*

$$\mathrm{g}_{\boldsymbol{\theta}}^{(k)}(\cdot) := \mathrm{GNN}_{\boldsymbol{\theta}}\big(\cdot, \mathbf{A}^{(k)}\big), \ k \in \{1, \ldots, K\}.$$

The intention of the *edge partitioner* and *community-GNN bank* is to capture the node information specific to each community. For instance, in a social network, such kind of information could be the different social roles of people when considered affiliated with different social groups, *e.g.*, one may be a former computer science major student in an alumni group, a research scientist in a work group, and an amateur chess player in a hobby club group. As shown in Section 4.3, the node embeddings produced by the *community-GNN bank* are separable by communities.

We finally introduce the *representation composer*, which embeds a combination of the information provided by each community into a global representation via a GNN:

Representation composer. *Let $\mathbf{H}^{(k)} := \mathrm{g}_{\boldsymbol{\theta}}^{(k)}(\mathbf{X}^*)$ denote the node representations learned from the kth community, where $k = 1, \ldots, K$, and $f(\cdot)$ denote the representation composer, whose functionality is to project a composite of community-specific node representations to one representation matrix,*

i.e., $\mathbf{H}_{\mathcal{V}} = f\left(\mathbf{H}^{(1)}, \mathbf{H}^{(2)}, \cdots, \mathbf{H}^{(K)}\right) := \mathrm{GNN}_{\boldsymbol{\theta}}\left(\|_{k=1}^{K} \mathbf{H}^{(k)}, \mathbf{A}\right)$. *We could further obtain* $\mathbf{h}_{\mathcal{G}}$*, the vector representation of graph* $\mathcal{G}$*, by sending the node level representations* $\mathbf{H}_{\mathcal{V}}$ *to a graph pooling layer, i.e.,* $\mathbf{h}_{\mathcal{G}} := \mathrm{GRAPHPOOL}(\mathbf{H}_{\mathcal{V}})$.

Taking `softmax` on the feature dimension of the *representation composer*'s output gives the predicted probabilities of labels, from which we are able to make predictions on the category of the unlabeled objects. Cascading the *edge partitioner* and *community-GNN bank* with the *representation composer* yields the probability $p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{X}, \mathbf{A}, \boldsymbol{\Phi})$.

2.3 Variational latent community inference

We use a *community encoder* as the module in the *inference network* to approximate the posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{y})$, and provide the *generative network* with the critical latent variable $\boldsymbol{\Phi}$:

Comunity encoder. *The community encoder models the variational posterior* $q_{\boldsymbol{\omega}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{X})$ *by a Weibull distribution with shape* $\mathbf{K}$ *and scale* $\boldsymbol{\Lambda}$*, whose parameters are learned by a GNN as* $\mathbf{K} \parallel \boldsymbol{\Lambda} = \mathrm{GNN}_{\boldsymbol{\omega}}\big(\mathbf{X}, \mathbf{A}\big)$. *A Weibull random sample from* $q_{\boldsymbol{\omega}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{X})$ *could be created through the inverse CDF transformation of a uniform random variable, given as follows:*

$$\boldsymbol{\Phi} = \boldsymbol{\Lambda} \odot \big(-\log(1-\mathbf{U})\big)^{(1 \oslash \mathbf{K})}, \mathrm{U}_{i,k} \overset{iid}{\sim} \mathrm{U}(0,1), \ \ \forall (i,k) \in \{1, \ldots, N\} \times \{1, \ldots, K\}, \tag{5}$$

where $\odot$ *and* $\oslash$ *denote element-wise multiplication and division.*

2.4 The overall training algorithm

We train VEPM by optimizing the evidence lower bound (ELBO), decomposed into three terms, as

$$\mathcal{L} = \mathcal{L}_{\mathrm{task}} + \mathcal{L}_{\mathrm{egen}} + \mathcal{L}_{\mathrm{KL}}, \tag{6}$$

where $\mathcal{L}_{\mathrm{task}} = \mathbb{E}_{q_{\boldsymbol{\omega}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{X})} \log p_{\boldsymbol{\theta}}(\mathbf{y}_o \mid \mathbf{A}, \boldsymbol{\Phi}, \mathbf{X})$, $\mathcal{L}_{\mathrm{egen}} = \mathbb{E}_{q_{\boldsymbol{\omega}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{X})} \log p_{\boldsymbol{\theta}}(\mathbf{A} \mid \boldsymbol{\Phi})$, and $\mathcal{L}_{\mathrm{KL}} = -D_{\mathrm{KL}}\big(q_{\boldsymbol{\omega}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{X}) \parallel p(\boldsymbol{\Phi})\big)$. These three terms correspond to the classification task, edge generation, and KL-regularization, respectively. Note that our specifications of $\boldsymbol{\Phi}$'s prior and variational posterior yield an analytical expression of $\mathcal{L}_{\mathrm{KL}}$, as described in detail in Appendix B.

Since modeling the likelihood of the labels involves stacking multiple modules embodied by deep neural networks, approximating the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{y}_o)$ is difficult from an optimization perspective. To effectively train VEPM, we leverage the edge generation process and break down the overall training pipeline into two phases: an *unsupervised pretrain*, followed by a *supervised finetune*. Specifically, we pretrain the *inference network* until convergence by optimizing $\mathcal{L}_{\mathrm{egen}} + \mathcal{L}_{\mathrm{KL}}$ in the first phase, after which we expect the inferred latent communities to be semantically meaningful to explain the generation of edges. We then join the pretrained *inference network* up with the *generative network*, and train the entire architecture with the objective function given in Equation (6). In the second phase, the label supervision is introduced by adding $\mathcal{L}_{\mathrm{task}}$, which finetunes the latent communities to align them with the end task. We summarize the overall training pipeline in Appendix A, and empirically study the effect of bi-phase training algorithm in Section 4.2.

3 Related Work

Generative semi-supervised graph deep learning. Advances in this line of work include embedding nodes with a deep graph encoder [7, 16], and training the embeddings by jointly reconstructing graph and optimizing a task-related objective, *e.g.*, semi-supervised node classification [17, 18] or community discovery [10, 11]. Similar to these models, we regularize the node embeddings $\boldsymbol{\Phi}$ by its probabilistic dependency with the graph structure, but instead of simply passing the learned embeddings to a black-box prediction network, we use them to develop a unique label generation process, employing community-specific GNNs for node representation learning and aggregation.

Modeling multi-relational data with GNNs. The topic of handling heterogeneous relational graphs has been extensively studied in the literature of mining Knowledge Graphs [19]. With the emergence of GNNs, quite a few attempts [20, 21] have been made to extend this graph learning paradigm to analytical tasks on multi-relational data. The general idea of adapting GNNs to this type of task is to individually process node representations aggregated from different types of relations, which is similar to our label generation process where information propagation in each community is also individually

modeled. However, for most of the instances of this type of application, the feature aggregation paths for different relations are separated either by annotated edge labels, or human-defined "edge types," whereas in our model, we partition the edges according to node interactions over latent communities, which are inferred from both the observed graph structure and labels from the end task.

Graph factorization based models. Previous works including DisenGCN [22] and FactorGCN [23] involve factorizing the observed graph and learning representations by combining representations of different graph factors. These methods expect to learn diverse graph factors that optimally facilitates the downstream tasks, and the graph factorization is driven purely from label supervision. By contrast, the factors we separate from the original graph is jointly determined by the task and a community-based graph generative model, which not only makes the factorization less dependent on label supervision, but also enhances model interpretability.

4 Empirical Evaluation

4.1 Node & graph classification

Datasets & experimental settings. For node classification, we consider three citation networks: Cora, Citeseer, and Pubmed, which provide bag-of-words document representations as node features and (undirected) citations as edges, and use the train/test/validation split standardized by Kipf and Welling [5]; For graph classification, we consider four bioinformatics datasets (MUTAG, PTC, NCI1, PROTEINS) and four social network datasets (IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY and REDDIT-MULTI). The input node features are crafted in the same way as Xu et al. [6]. Most of the baselines are compared following the 10-fold cross validation based evaluation protocol proposed by Xu et al. [6]. Since this protocol is criticized for being prone to overestimating model performance [24], we also evaluate our model following Zhang and Chen [25], who conduct a more rigorous train-validation-test protocol. To counter the memory complexity expansion by using K GNNs in the *community-GNN bank*, we limit the size of hidden dimension in each *community-GNN* to $1/K$ of what is commonly used among the baselines (*e.g.*, 64 for both tasks). We elaborate other details of the benchmarks, model architecture, and hyperparameter determination in Appendix C.

Node classification. We use classification accuracy as the evaluation metric for node classification. Table 1 reports the average performance of VEPM ($\pm$ standard error) against related baselines that are categorized into three different groups. The first group consists of the GCNs [5] and its variant GCN-64 that expands the hidden dimension from 16 to 64. VEPM outperforms the first group by a significant margin. The explanation is twofold: (i) VEPM augments the node attributes with community information carried by the inferred node-community affiliations, (ii) VEPM learns community-specific node embeddings. The second group includes SIG-VAE [17] and WGCAE [18], both of which learn node embeddings from graph generative models jointly optimized with a supervised loss. The performance gain that VEPM obtains could be attributed to our unique label generation process that integrates community information into not only node attributes, but also neighborhood aggregation. The third group [13, 26, 22] is focused on learning node embeddings leveraging heterogeneous hidden relations. When fitting the hidden relations via attention mechanism, they only use information from the node features through label supervision, whereas our approach also takes the observed graph structure into account via a graph generative model. The additional information from the graph utilized by VEPM could explain the enhancement achieved by VEPM over the third group.

Table 1: Comparison of node classification performance. The asterisk ($*$) marks the results reported by Veličković et al. [13].

Method	Cora	Citeseer	Pubmed
ChebNet [4]	81.2	69.8	74.4
GCN [5]	81.5	70.3	79.0
GCN-64*	81.4	70.9	79.0
SIG-VAE [17]	79.7	70.4	79.3
WGCAE [18]	82.0	72.1	79.1
GAT* [13]	83.0	72.5	79.0
hGANet [26]	83.5	72.7	79.2
DisenGCN [22]	83.7	**73.4**	80.5
VEPM (this work)	**84.3** $\pm$ 0.1	72.5 $\pm$ 0.1	**82.4** $\pm$ 0.2

Graph classification. For the first protocol, we compare VEPM with classical graph classification baselines [27–29], generic-GNN-based models [13, 26, 6], and GNNs with relation-based or task-driven graph factorization [20, 21, 23]. Results in Table 2 show that VEPM achieves state-of-the-art graph classification performance on 5 out of 8 benchmarks, and the second-best performance on the other 3 benchmarks, including NCI1, where VEPM outperforms all the other GNN-based models.

Table 2: Comparison of graph classification performance (average accuracy ± standard error).

Method	IMDB-B	IMDB-M	MUTAG	PTC	NCI1	PROTEINS	RDT-B	RDT-M
WL subtree [27]	73.8 ± 3.9	50.9 ± 3.8	90.4 ± 5.7	59.9 ± 4.3	**86.0** ± 1.8	75.0 ± 3.1	81.0 ± 3.1	52.5 ± 2.1
PATCHYSAN [28]	71.0 ± 2.2	45.2 ± 2.8	92.6 ± 4.2	60.0 ± 4.8	78.6 ± 1.9	75.9 ± 2.8	86.3 ± 1.6	49.1 ± 0.7
AWE [29]	74.5 ± 5.9	51.5 ± 3.6	87.9 ± 9.8	-	-	-	87.9 ± 2.5	54.7 ± 2.9
GAT [13]	70.5 ± 2.3	47.8 ± 3.1	89.4 ± 6.1	66.7 ± 5.1	-	-	-	-
hGANet [26]	-	49.0	90.0	65.0	-	78.7	-	-
GIN [6]	75.1 ± 5.1	52.3 ± 2.8	89.4 ± 5.6	64.6 ± 7.0	82.7 ± 1.6	76.2 ± 2.8	**92.4** ± 2.5	**57.5** ± 1.5
R-GCN [20]	-	-	82.3 ± 9.2	67.8 ± 13.2	-	-	-	-
CompGCN [21]	-	-	89.0 ± 11.1	71.6 ± 12.0	-	-	-	-
FactorGCN [23]	75.3 ± 2.7	-	89.9 ± 6.5	-	-	-	-	-
VEPM (this work)	**76.7** ± 3.1	**54.1** ± 2.1	**93.6** ± 3.4	**75.6** ± 5.9	83.9 ± 1.8	**80.5** ± 2.8	90.5 ± 1.8	55.0 ± 1.5

For the second protocol, aside from a traditional method, WL [27], and a generic-GNN-based method, DGCNN [30], we compare VEPM with two GNNs [31, 25] that adapt capsule neural networks [32] to graph-structured data, which aim to learn different aspects of graph properties via dynamic routing [33]. The results in Table 3 show that VEPM outperforms these GNN-based methods on most of the benchmarks. This indicates that the aspects of graph information, as defined with communities and learned via aggregating node features with partitioned graphs, are more pertinent to the end tasks.

Table 3: Graph classification accuracies under the protocol by Zhang and Chen [25].

Method	IMDB-B	IMDB-M	MUTAG	NCI1	PROTEINS	RDT-M
WL subtree [27]	73.4 ±4.6	49.3 ±4.8	82.1 ±0.4	82.2 ±0.2	74.7 ±0.5	49.4 ±2.4
DGCNN [30]	70.0 ±0.9	47.8 ±0.9	85.8 ±1.7	74.4 ±0.5	75.5 ±0.9	48.7 ±4.5
GCAPS-CNN [31]	71.7 ±3.4	48.5 ±4.1	-	**82.7** ±2.4	76.4 ±4.2	50.1 ±1.7
CapsGNN [25]	73.1 ±4.8	50.3 ±2.7	86.7 ±6.9	78.4 ±1.6	76.3 ±3.6	**52.9** ±1.5
VEPM (this work)	**74.6**±2.1	**50.6** ±3.6	**91.5** ± 4.3	81.7 ± 1.4	**76.8** ± 4.1	50.2 ± 2.1

Classification with reduced labels. The gain of incorporating the graph generation process into VEPM is that the observed graph structure also provides information for hidden community detection and edge partition, which is beneficial if the amount of label is not sufficient for both decomposing the graph into hidden factors and semi-supervised task-learning. To illustrate this point, based on the evaluation protocol by Kipf and Welling [5] for node classification and Xu et al. [6] for graph classification, we re-evaluate the performance of VEPM under reduced training labels on Cora (for node classification) and MUTAG (for graph classification), along with generic-GNN-based methods [5, 13, 6] and decomposition-based methods [22, 23]. The results are shown in Figure 2: performance of all models are negatively impacted by the reduction in training labels, those suffering from the strongest performance decline are DisenGCN [22] and FactorGCN [23] that decompose graphs solely based on label supervision, while VEPM also performs graph factorization with limited labels, it consistently outperforms the other methods under all settings, and the superiority becomes even more evident as the amount of annotated data further decreases. We consider the enhanced robustness under sparsely labeled data as a crucial feature to have for real-world graphs.

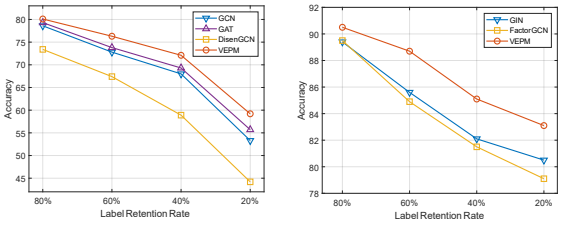


Figure 2: Node classification accuracy on Cora (left) and graph classification accuracy on MUTAG (right) as the number of labels decreases. The horizontal axis is the keep rate of labels in the training set under corresponding protocols.

4.2 Ablation studies

In this part, we explore the alternative choices of the input features and training scheme, from which we highlight the benefits of the current design of VEPM.

Various input features. We evaluate the model performance with four types of inputs: random noise, hand-crafted features by Xu et al. [6], node-community affiliation scores (*i.e.*, $\mathbf{\Phi}$), and the combination of the last two. The quality of these inputs are evaluated by the graph classification accuracy as summarized in Table 4. Two conclusions could be drawn: (i) $\mathbf{\Phi}$ is comparably informative to the end task as hand-crafted node features, both are significantly better than random noise (see rows 1 – 3 of Table 4); (ii) the information $\mathbf{\Phi}$ and $\mathbf{X}$ hold for the task enhances the discriminative

level of learned representations in a complementary manner (see rows 2, 4 of Table 4). These results verify the quality of $\mathbf{\Phi}$ inferred by VEPM as additional node features.

Table 4: Comparison of VEPMs with various input node features.

Random	Hand-crafted	Community-based	IMDB-B	IMDB-M	MUTAG	PTC	NCI1	PROTEINS
✔			64.7 ± 1.6	42.3 ± 1.5	84.6 ± 4.3	63.6 ± 2.0	67.5 ± 2.1	76.5 ± 3.5
	✔		$\mathbf{80.3} \pm 2.0$	53.5 ± 2.6	93.1 ± 5.0	74.7 ± 4.1	68.0 ± 2.1	77.9 ± 2.7
		✔	74.7 ± 5.1	51.5 ± 2.0	84.6 ± 5.5	68.9 ± 3.9	80.8 ± 1.4	75.7 ± 3.8
	✔	✔	76.7 ± 3.1	$\mathbf{54.1} \pm 2.1$	$\mathbf{93.6} \pm 3.5$	$\mathbf{75.6} \pm 5.9$	$\mathbf{83.9} \pm 1.8$	$\mathbf{80.5} \pm 2.8$

With or without pretraining. Finally, we compare the performance of VEPM that undergoes both pretraining and finetuning against that of a VEPM trained with Equation (6) from scratch. Results in Table 5 shows that comparing with training from scratch, the state of the *community encoder* after pretraining is better for the classification tasks, suggesting the benefits brought by pretraining.

Table 5: Comparison of VEPMs with different training schemes.

	IMDB-B	IMDB-M	MUTAG	PTC	NCI1	PROTEINS
trained from scratch	61.0 ± 3.6	46.8 ± 0.8	86.6 ± 6.6	68.6 ± 2.0	67.2 ± 8.4	60.6 ± 4.1
pretrain + finetune	$\mathbf{76.7} \pm 3.1$	$\mathbf{54.1} \pm 2.1$	$\mathbf{93.6} \pm 3.5$	$\mathbf{75.6} \pm 5.9$	$\mathbf{83.9} \pm 1.8$	$\mathbf{80.5} \pm 2.8$

4.3 Qualitative analysis

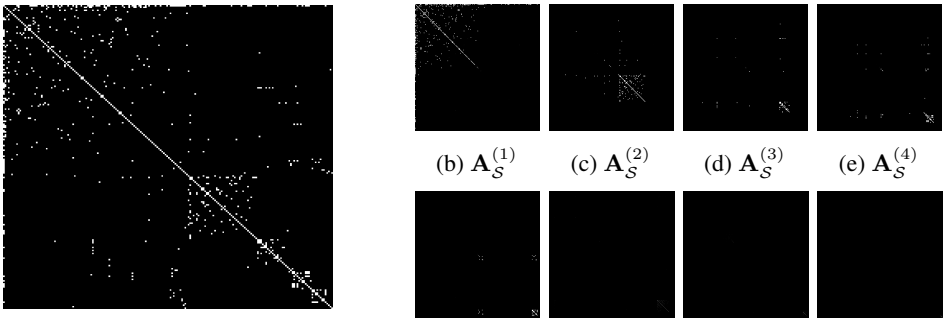
Visualizing community structures. To show that through partitioning the edges, VEPM identifies each latent community from the original graph, in Figure 3, we plot the adjacency matrices of a 200-node subgraph of Cora, before and after edge partition. The subgraph is created via breadth-first search [34] node selection to ensure connectivity. We sort the nodes sample $\mathbb{S}$ in order to present a clearer view of the community structures. Specifically, we prepare K buckets; for node u, we compute the total interactions it engages under metacommunity k by $\mu_{u,k} := \sum_{v \in \mathbb{S}} \mathbf{\Phi}_{u,k} \mathrm{diag}(\boldsymbol{\gamma}_k) \mathbf{\Phi}'_{v,k}$, and assign it to bucket k', where $k' = \arg\max_k \mu_{u,k}$. We first sort the buckets by descending their counts of assigned nodes, then sort the nodes within bucket $k,\ k \in \{1, \ldots, K\}$, by descending value of $\mu_{u,k}$. The $\mathbf{\Phi}$ used for sorting is sampled at the end of training.

The community structures that VEPM detects could be found in Figure 3a as the blocks on the main diagonal. The fact that most of the bright spots (*i.e.*, edges) are located in these on-diagonal blocks reflects dense node connections internal to each community. Sparsely distributed off-diagonal spots provide evidence of overlapping membership between the communities, which is also taken into account by our graph generative model. Figures 3b to 3i show that the *edge partitioner* has done a sound job in separating latent communities. There are metacommunities in which not a single pair of nodes has strong interaction, which accounts for the appeared emptiness in Figures 3h and 3i. Such kind of artifact disappears when replacing the subgraph by the full graph of Cora, as visualized in Figures 5 to 13 in Appendix G.

Visualizing latent representations. The previous visualization experiment verifies the following propositions: (i) the identified hidden structures are latent communities, and (ii) the partitioned graphs are different from each other. We now study how they benefit representation learning. Cora and MUTAG are selected as the representatives for node and graph classification benchmarks. For the MUTAG dataset, we remove graphs that contain node categories with less than 5 instances (less than 10 from a totality of 188) and randomly sample 10 graphs for the visualization experiments.

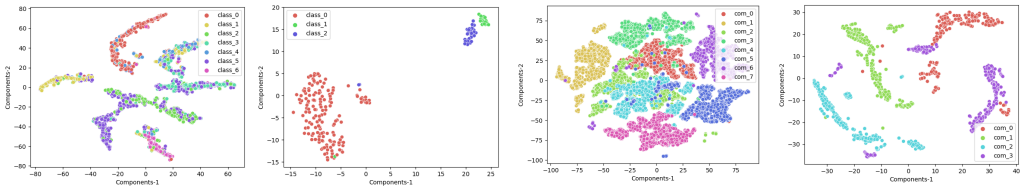
We first visualize $\mathbf{\Phi}$ obtained at the end of the *unsupervised pretrain* stage (Figures 4a and 4b), where the node-community affiliation vectors are projected to 2-D space via t-SNE [35]. Proposition (i) ensures that the node information carried by $\mathbf{\Phi}$ is about communities. We color code the scatters by node labels, both Figures 4a and 4b exhibit strong correlation between the spatial clusters and colors, which indicates that even without label supervision, the node information provided with $\mathbf{\Phi}$ has discriminative power on classifying nodes, which is compatible with the conclusions in Section 4.2.

We then visualize the community-specific node emebddings obtained at the end of the *supervised finetune* stage (Figures 4c and 4d). Similarly, t-SNE is adopted to reduce the dimensionality of obtained node embeddings. This time we color code the scatters by the latent metacommunity they correspond to. Both Figures 4c and 4d show clear boundaries between the colored clusters, which shows that each community enriches node information from a nonrepetitive angle, potentially



(a) $\mathbf{A}_{\mathcal{S}}$ (b) $\mathbf{A}_{\mathcal{S}}^{(1)}$ (c) $\mathbf{A}_{\mathcal{S}}^{(2)}$ (d) $\mathbf{A}_{\mathcal{S}}^{(3)}$ (e) $\mathbf{A}_{\mathcal{S}}^{(4)}$ (f) $\mathbf{A}_{\mathcal{S}}^{(5)}$ (g) $\mathbf{A}_{\mathcal{S}}^{(6)}$ (h) $\mathbf{A}_{\mathcal{S}}^{(7)}$ (i) $\mathbf{A}_{\mathcal{S}}^{(8)}$

Figure 3: S is a connected subgraph of 200 nodes sampled from Cora, whose adjacency matrix $\mathbf{A}_{\mathcal{S}}$ is visualized in (a). (b)–(i) show the results of edge partition, the notation $\mathbf{A}_{\mathcal{S}}^{(k)}$ denotes the kth output from the *edge partitioner*. Brighter color represents larger edge weights.



(a) $\mathbf{\Phi}$, Cora (b) $\mathbf{\Phi}$, MUTAG (c) $\mathrm{g}_{\boldsymbol{\theta}}^{(\cdot)}(\mathbf{X}^*)$, Cora (d) $\mathrm{g}_{\boldsymbol{\theta}}^{(\cdot)}(\mathbf{X}^*)$, MUTAG

Figure 4: t-SNE visualization of $\mathbf{\Phi}$ obtained from *unsupervised pretrain* on (a) Cora and (b) MUTAG, as well as community-specific node embeddings obtained during *supervised finetune* on (c) Cora and (d) MUTAG.

enhancing the overall discriminative power of the ensembeled node or graph representations and leading to better performance on downstream tasks.

4.4 Complexity analysis

Time complexity. The time complexity of (pre)training VEPM is $O(N^2)$ for modeling the connection statuses of all $\binom{N}{2}$ node pairs, and could be reduced to $O(N_s^2)$ if subgraph-sampling based acceleration is applied. Here N_s denotes the number of nodes in the subgraph. Effects and implications of adopting such type of acceleration are discussed in Appendix E. The evaluation time complexity of VEPM is comparable to a (2-layer-GCN)+GAT for node classification, and a (2-layer-GCN)+GIN for graph classification.

Space complexity. Supposing the input and output dimensions are both H, a GNN layer would have H^2 parameters, given the memory reduction measure (as described in Section 4.1) we take, each layer in one *community-GNN* has either H^2/K parameters if the input is shared or H^2/K^2 if not. With K *community-GNN*s, the number of parameters in one layer is between $[H^2/K, H^2]$. Although it reduces certain amount of space complexity comparing with a regular GNN, there are overheads added by the *community encoder* and hence the overall space complexity of VEPM is comparable with baselines such as GCN or GIN. For node and graph classification, it is comparable to a 4-layer-GCN and a (2-layer-GCN)+GIN, respectively.

5 Conclusion

Moving beyond treating the graph adjacency matrix as given, we develop variational edge partition models (VEPMs) to extract overlapping node communities and perform community-specific node feature aggregations. Specifically, we first utilize a GNN-based inference network to obtain node-community affiliation strengths, with which we augment node attributes, and partition the edges according to the intensities of node interactions with respect to each community. We learn GNN-based node embeddings for each community by aggregating node features with the corresponding partitioned graph, and aggregate all community-specific node embeddings for the downstream tasks. Extensive qualitative and quantitative experiments on both node-level and graph-level classification tasks are performed to illustrate the working mechanism and demonstrate the efficacy of VEPMs in supervised graph representation learning.

References

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[2] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[3] James Atwood and Don Towsley. Diffusion-convolutional neural networks. volume 29, 2016.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[6] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[7] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[8] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[9] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[10] Fan-Yun Sun, Meng Qu, Jordan Hoffmann, Chin-Wei Huang, and Jian Tang. vgraph: A generative model for joint community detection and node representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[11] Nikhil Mehta, Lawrence Carin, and Piyush Rai. Stochastic blockmodels meet graph neural networks. In *International Conference on Machine Learning (ICML)*, pages 4466–4474, 2019.

[12] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations (ICLR)*, 2020.

[13] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[14] Edo M Airoldi, David Blei, Stephen Fienberg, and Eric Xing. Mixed membership stochastic blockmodels. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 21, 2009.

[15] Mingyuan Zhou. Infinite edge partition models for overlapping community detection and link prediction. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 38 of *Proceedings of Machine Learning Research*, pages 1135–1143, 2015.

[16] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[17] Arman Hasanzadeh, Ehsan Hajiramezanali, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Semi-implicit graph variational auto-encoders. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 10711–10722, 2019.

[18] Chaojie Wang, Hao Zhang, Bo Chen, Dongsheng Wang, Zhengjue Wang, and Mingyuan Zhou. Deep relational topic modeling via graph poisson gamma belief network. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

[19] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[20] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference (ESWC)*, pages 593–607. Springer, 2018.

[21] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[22] Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. Disentangled graph convolutional networks. In *International Conference on Machine Learning (ICML)*, pages 4212–4221, 2019.

[23] Yiding Yang, Zunlei Feng, Mingli Song, and Xinchao Wang. Factorizable graph convolutional networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 20286–20296, 2020.

[24] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations*, 2020.

[25] Xinyi Zhang and Lihui Chen. Capsule graph neural network. In *International Conference on Learning Representations (ICLR)*, 2019.

[26] Hongyang Gao and Shuiwang Ji. Graph representation learning via hard and channel-wise attention networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 741–749, 2019.

[27] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[28] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning (ICML)*, pages 2014–2023, 2016.

[29] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *International conference on machine learning (ICML)*, pages 2186–2195, 2018.

[30] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[31] Saurabh Verma and Zhi-Li Zhang. Graph capsule convolutional neural networks. *arXiv preprint arXiv:1805.08090*, 2018.

[32] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer, 2011.

[33] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[34] Edward Forrest Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

[35] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[36] Hao Zhang, Bo Chen, Dandan Guo, and Mingyuan Zhou. WHAI: Weibull hybrid autoencoding inference for deep topic modeling. In *International Conference on Learning Representations (ICLR)*, 2018.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Abstract
   (b) Did you describe the limitations of your work? [Yes] See Appendix
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1 and Appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 4
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See supplemental material
   (b) Did you mention the license of the assets? [Yes] See supplemental material
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See supplemental material
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]