

Accurate Node Feature Estimation with Structured Variational Graph Autoencoder

Jaemin Yoo*
Carnegie Mellon University
Pittsburgh, PA, USA
jaeminyoo@cmu.edu

Hyunsik Jeon
Seoul National University
Seoul, South Korea
jeon185@snu.ac.kr

Jinhong Jung
JBNU
Jeonju, South Korea
jinhongjung@jbnu.ac.kr

U Kang
Seoul National University
Seoul, South Korea
ukang@snu.ac.kr

ABSTRACT

Given a graph with partial observations of node features, how can we estimate the missing features accurately? Feature estimation is a crucial problem for analyzing real-world graphs whose features are commonly missing during the data collection process. Accurate estimation not only provides diverse information of nodes but also supports the inference of graph neural networks that require the full observation of node features. However, designing an effective approach for estimating high-dimensional features is challenging, since it requires an estimator to have large representation power, increasing the risk of overfitting. In this work, we propose SVGA (Structured Variational Graph Autoencoder), an accurate method for feature estimation. SVGA applies strong regularization to the distribution of latent variables by structured variational inference, which models the prior of variables as Gaussian Markov random field based on the graph structure. As a result, SVGA combines the advantages of probabilistic inference and graph neural networks, achieving state-of-the-art performance in real datasets.

CCS CONCEPTS

• **Information systems** → *Social networks*; • **Computing methodologies** → *Learning in probabilistic graphical models*.

KEYWORDS

feature estimation, graph neural networks, variational inference

ACM Reference Format:

Jaemin Yoo, Hyunsik Jeon, Jinhong Jung, and U Kang. 2022. Accurate Node Feature Estimation with Structured Variational Graph Autoencoder. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22)*, August 14–18, 2022, Washington, DC, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3534678.3539337>

1 INTRODUCTION

Given a graph with partial observations of node features, how can we estimate the missing features accurately? Many real-world data are represented as graphs to model the relationships between entities.

*This work was done when the author was at Seoul National University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD '22, August 14–18, 2022, Washington, DC, USA.

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9385-0/22/08...\$15.00

<https://doi.org/10.1145/3534678.3539337>

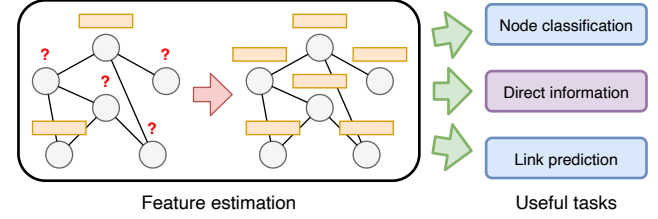


Figure 1: An illustration of the feature estimation problem. The generated features not only provide direct information of node properties but also help other graph-related tasks.

Social networks, seller-item graphs in electronic commerce, and user-movie graphs in a streaming service are all examples of graph data that have been studied widely in literature [12, 18, 23, 26, 33]. Such graphs become more powerful when combined with feature vectors that describe the diverse properties of nodes [6, 29].

However, node features are commonly missing in a real-world graph. Users in an online social network set their profiles private, and sellers in electronic commerce often register items without an informative description. In such cases, even the observed features cannot be used properly due to the missing ones, since many graph algorithms assume the full observation of node features. Figure 1 illustrates the feature estimation problem in an example graph. An accurate estimation of missing features not only provides diverse information of node properties but also improves the performance of essential tasks such as node classification or link prediction by providing important evidence for training a classifier.

However, accurate estimation of missing features is challenging due to the following reasons. First, target nodes have no specific information that describes their properties. The main evidence for estimation is the graph structure, which gives only partial information of nodes based on the relationships with the other nodes. Second, the target variables are high-dimensional vectors containing up to thousands of elements. This requires large representation power for accurate estimation, involving a high risk of overfitting as a consequence. Existing approaches [2, 3, 13] failed to address such challenges effectively, resulting in limited performance.

We propose SVGA (Structured Variational Graph Autoencoder), an accurate method for missing feature estimation. The main idea for addressing the challenges is to run *structured* variational inference to effectively regularize the distribution of latent variables by modeling their correlations from the structure of a graph. We first propose stochastic inference, which models the prior of latent variables as Gaussian Markov random field (GMRF). Then, we improve the stability of inference with our proposed deterministic modeling, which results in a new graph-based regularizer. These allow us to

avoid the overfitting without degrading the representation power, achieving state-of-the-art performance in real-world datasets.

Our contributions are summarized as follows:

- **Method.** We propose SVGA, an accurate method for missing feature estimation. SVGA introduces a new way to run variational inference on graph-structured data with modeling the correlations between target variables as GMRF.
- **Theory.** We analyze the theoretical properties of structured variational inference with the stochastic and deterministic modeling. We also analyze the time and space complexities of our SVGA, which are both linear with the number of nodes and edges of a given graph, showing its scalability.
- **Experiments.** Extensive experiments on eight real-world datasets show that SVGA provides state-of-the-art performance with up to 16.3% higher recall and 14.0% higher nDCG scores in feature estimation, and up to 10.4% higher accuracy in node classification compared to the best competitors.

The rest of this paper is organized as follows. In Section 2, we introduce the problem definition and preliminaries of SVGA. In Section 3, we propose SVGA and discuss its theoretical properties. We present experimental results in Section 4 and describe related works in Section 5. We conclude in Section 6. The code and datasets are available at <https://github.com/snudatalab/SVGA>.

2 PRELIMINARIES

We introduce the problem definition and preliminaries, including Gaussian Markov random field and variational inference.

2.1 Missing Feature Estimation

The feature estimation problem is defined as follows. We have an undirected graph $G = (\mathcal{V}, \mathcal{E})$, where \mathcal{V} and \mathcal{E} represent the sets of nodes and edges, respectively. A feature vector \mathbf{x}_i exists for every node i , but is observable only for a subset $\mathcal{V}_x \subset \mathcal{V}$ of nodes. Our goal is to predict the missing features of *test* nodes $\mathcal{V} \setminus \mathcal{V}_x$ using the structure of G and the observations for \mathcal{V}_x . The problem differs from generative learning [16] in that there exist correct answers; generative learning is typically an unsupervised problem.

We also assume that the label y_i of each node i can be given as an additional input for a set \mathcal{V}_y of nodes such that $\mathcal{V}_y \subseteq \mathcal{V}$. Such labels improve the accuracy of feature estimation, especially when they provide information for the test nodes: $\mathcal{V}_y \cap (\mathcal{V} \setminus \mathcal{V}_x) \neq \emptyset$. This is based on the idea that categorical labels are often easier to acquire than high-dimensional features, and knowing the labels of target nodes gives a meaningful advantage for estimation. Thus, we design our framework to be able to work with $\mathcal{V}_y \neq \emptyset$, although we consider $\mathcal{V}_y = \emptyset$ as a base setup of experiments for the consistency with previous approaches that take only the observed features.

2.2 Gaussian Markov Random Field

Gaussian Markov random field (GMRF) [19] is a graphical model that represents a multivariate Gaussian distribution. Given a graph $G = (\mathcal{V}, \mathcal{E})$ whose nodes have continuous signals that are correlated by the graph structure, GMRF represents the distribution of signals with two kinds of potential functions ψ_i and ψ_{ij} for every node i and edge (i, j) , respectively. We assume the signal of each node i as a random variable Z_i with a possible value z_i .

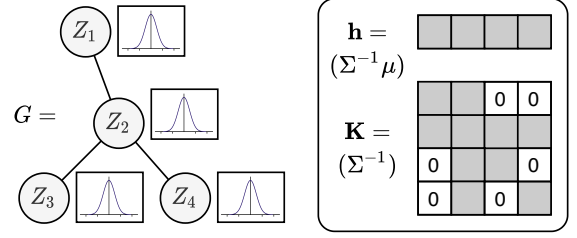


Figure 2: Gaussian Markov random field (GMRF) describing a Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ by parameters \mathbf{h} and \mathbf{K} . The nonzero entries in \mathbf{K} correspond to the edges in G .

Specifically, the node potential ψ_i for each node i and the edge potential ψ_{ij} for each edge (i, j) are defined as follows:

$$\psi_i(z_i) = \exp(-0.5K_{ii}z_i^2 + h_iz_i) \quad (1)$$

$$\psi_{ij}(z_i, z_j) = \exp(-K_{ij}z_iz_j), \quad (2)$$

where $\mathbf{h} \in \mathbb{R}^n$ and $\mathbf{K} \in \mathbb{R}^{n \times n}$ are the parameters of the GMRF, and n is the number of nodes. The nonzero elements of \mathbf{K} correspond to the edges of the graph as depicted in Figure 2.

Then, the joint probability $p(\mathbf{z})$ is given as the multiplication of all potential functions:

$$p(\mathbf{z}) = \frac{1}{C} \prod_{i \in \mathcal{V}} \psi_i(z_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(z_i, z_j), \quad (3)$$

where C is a normalization constant. Each potential measures how likely z_i or (z_i, z_j) appears with the current probabilistic assumption with the parameters \mathbf{h} and \mathbf{K} , and the joint probability is computed by multiplying the potentials for all nodes and edges.

The roles of parameters \mathbf{K} and \mathbf{h} can be understood with respect to the distribution that GMRF represents. Lemma 2.1 shows that GMRF is equivalent to a multivariate Gaussian distribution whose mean and covariance are determined by \mathbf{K} and \mathbf{h} . \mathbf{K} is the inverse of the covariance Σ , and a pair of signals z_i and z_j is more likely to be observed if K_{ij} is small. \mathbf{h} determines the mean of the signals if \mathbf{K} is fixed, and is typically set to zero as we assume no initial bias of signals for the simplicity of computation.

LEMMA 2.1. *The joint probability of Equation (3) is the same as the probability density function of a multivariate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$, where $\mu = \mathbf{K}^{-1}\mathbf{h}$ and $\Sigma = \mathbf{K}^{-1}$.*

PROOF. See Appendix A.1. \square

We utilize GMRF to incorporate a real-world graph in a probabilistic framework. Specifically, we generate a multivariate Gaussian distribution that models the probabilistic relationships between nodes by designing \mathbf{K} and \mathbf{h} from the adjacency matrix \mathbf{A} of the given graph. GMRF plays a crucial role in our proposed approach, which aims to run variational inference in graph-structured data without ignoring the correlations between target variables.

2.3 Variational Inference for Joint Learning

Variational inference [16, 17, 25] is a technique for approximating intractable posterior distributions, which has been used widely for generative learning. Given the adjacency matrix \mathbf{A} of a graph, our goal is to find optimal parameters Θ that maximize the likelihood $p_\Theta(\mathbf{X}, \mathbf{y} \mid \mathbf{A})$ of observed features \mathbf{X} and labels \mathbf{y} . We introduce a

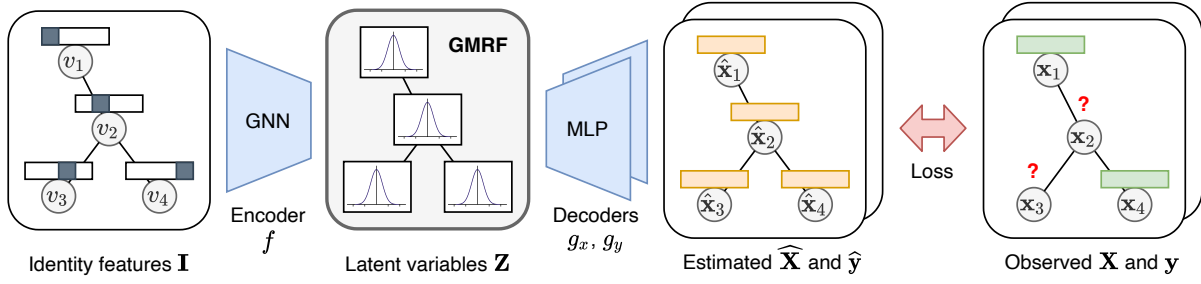


Figure 3: The structure of our SVGA, which consists of an encoder network f and two decoder networks g_x and g_y for features and labels, respectively. We model the distribution of latent variables with GMRF, exploiting the graph structure for modeling the correlations between target variables. The label decoder g_y works as an auxiliary module that helps g_x .

latent variable $\mathbf{z}_i \in \mathbb{R}^d$ for each node i and denote the realization of all latent variables by $\mathbf{Z} \in \mathbb{R}^{n \times d}$, where n is the number of nodes and d is the size of variables. The latent variable \mathbf{z}_i represents the characteristic of each node i for estimating its feature \mathbf{x}_i .

With variational inference, we change the problem into maximizing the evidence lower bound (ELBO):

$$\begin{aligned} \log p_{\theta}(\mathbf{X}, \mathbf{y} \mid \mathbf{A}) &\geq \mathcal{L}(\Theta) \\ &= \mathbb{E}_{\mathbf{Z} \sim q_{\phi}(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})} [\log p_{\theta, \rho}(\mathbf{X}, \mathbf{y} \mid \mathbf{Z}, \mathbf{A})] \\ &\quad - D_{\text{KL}}(q_{\phi}(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A}) \parallel p(\mathbf{Z} \mid \mathbf{A})), \end{aligned} \quad (4)$$

where $\mathcal{L}(\Theta)$ is the ELBO, q_{ϕ} is a parameterized distribution of \mathbf{Z} , and $p_{\theta, \rho}$ is a parameterized distribution of \mathbf{X} and \mathbf{y} . The first term of $\mathcal{L}(\Theta)$ is the likelihood of observed variables given \mathbf{Z} , while the second term measures the difference between $q_{\phi}(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$ and the prior distribution $p(\mathbf{Z} \mid \mathbf{A})$ by the KL divergence.

We assume the conditional independence between \mathbf{X} , \mathbf{y} , and \mathbf{A} given \mathbf{Z} , expecting that each variable \mathbf{z}_i has sufficient information of node i to generate its feature \mathbf{x}_i and label y_i . Then, the first term of $\mathcal{L}(\Theta)$ in Equation (4) is rewritten as follows:

$$\begin{aligned} &\mathbb{E}_{\mathbf{Z} \sim q_{\phi}(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})} [\log p_{\theta, \rho}(\mathbf{X}, \mathbf{y} \mid \mathbf{Z}, \mathbf{A})] \\ &= \mathbb{E}_{\mathbf{Z} \sim q_{\phi}(\mathbf{Z} \mid \cdot)} \left[\sum_{i \in \mathcal{V}_x} \log p_{\theta}(\mathbf{x}_i \mid \mathbf{z}_i) + \sum_{i \in \mathcal{V}_y} \log p_{\rho}(y_i \mid \mathbf{z}_i) \right], \end{aligned} \quad (5)$$

where \mathcal{V}_x and \mathcal{V}_y are the sets of nodes whose features and labels are observed, respectively, and $q_{\phi}(\mathbf{Z} \mid \cdot)$ denotes $q_{\phi}(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$.

Equation (5) represents the conditional likelihood of observed features and labels given \mathbf{Z} . Thus, maximizing Equation (5) is the same as minimizing the reconstruction error of observed variables in typical autoencoders. On the other hand, the KL divergence term in Equation (4) works as a regularizer that forces the distribution $q_{\phi}(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$ of latent variables to be close to the prior $p(\mathbf{Z} \mid \mathbf{A})$. The characteristic of regularization depends on how we model the prior $p(\mathbf{Z} \mid \mathbf{A})$, which plays an essential role in our framework.

Note that the objective function of Equation (4) works whether the observed labels \mathbf{y} are given or not, due to our assumption on the conditional independence between \mathbf{X} and \mathbf{y} . Only the first term of Equation (5) is used if there are no observed labels.

3 PROPOSED METHOD

We propose SVGA (Structured Variational Graph Autoencoder), an accurate method for missing feature estimation. The main ideas of SVGA are summarized as follows:

- **GNN with identity node features (Sec. 3.1).** We address the deficiency of input features by utilizing a graph neural network (GNN) with identity node features as an encoder function, which allows us to learn an independent embedding vector for each node during the training.
- **Structured variational inference (Sec. 3.2).** We propose a new way to run variational inference on graph-structured data without ignoring the correlations between target examples. This is done by modeling the prior distribution of latent variables with Gaussian Markov random field (GMRF).
- **Unified deterministic modeling (Sec. 3.3).** We improve the stability of inference by changing the stochastic sampling of latent variables into a deterministic process. This makes the KL divergence term of ELBO as a general regularizer that controls the space of node representations.

In Section 3.1, we introduce the overall structure of SVGA and the objective function for its training. Then in Sections 3.2 and 3.3, we induce our graph-based regularizer from structured variational inference. Specifically, we propose the basic parameterization of structured inference in Section 3.2 and improve its stability with the deterministic modeling of latent variables in Section 3.3.

3.1 Overall Structure of SVGA

Figure 3 shows the overall structure of SVGA, which consists of an encoder f and two decoder networks g_x and g_y . The networks f , g_x and g_y are designed to estimate the target distributions of the ELBO of Equation (4): $q_{\phi}(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$, $p_{\theta}(\mathbf{x}_i \mid \mathbf{z}_i)$ and $p_{\rho}(y_i \mid \mathbf{z}_i)$, respectively, where ϕ , θ , and ρ are their parameters. The encoder f generates latent representations of nodes, and the decoders g_x and g_y use the generated representations to estimate the features and labels of nodes. The feature decoder g_x makes the final estimation of missing features, while the label decoder g_y helps the training of g_x and is not used if no labels are observed.

3.1.1 Encoder Network. The encoder network f aims to model the latent distribution $q_{\phi}(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$ with parameters ϕ . We propose to use a graph neural network (GNN) as f , because the main functionality required for f is to generate an embedding vector for each node following the graphical structure. In experiments, we adopt a simple graph convolutional network (GCN) [18] as f , which works well even when the amount of training data is insufficient.

Still, it is required that every node contains a feature vector to run the GNN encoder on the given graph. Only a few nodes have

Algorithm 1 Training of SVGA with deterministic inference.

Input: Adjacency matrix \mathbf{A} , diagonal adjacency \mathbf{D} , feature \mathbf{X} , (optional) one-hot label \mathbf{Y} , hyperparameters α , β and λ , networks f , g_x , and g_y , and their parameters ϕ , θ , and ρ , respectively

Output: Updated parameters ϕ' , θ' , and ρ'

- 1: $\mathbf{Z} \leftarrow \mathbf{E} \leftarrow f(\mathbf{A}; \phi)$ ▷ Run the unified encoder
 - 2: $\hat{\mathbf{X}}, \hat{\mathbf{Y}} \leftarrow g_x(\mathbf{Z}, \mathbf{A}; \theta), g_y(\mathbf{Z}, \mathbf{A}; \rho)$ ▷ Make predictions
 - 3: $l_{xy} \leftarrow \sum_i l_x(\hat{\mathbf{x}}_i, \mathbf{x}_i) + \sum_j l_y(\hat{\mathbf{y}}_j, \mathbf{y}_j)$ ▷ Equation (7) to (10)
 - 4: $\mathbf{K} \leftarrow \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$ ▷ Equation (11)
 - 5: $l_{\text{GMRF}} \leftarrow \text{tr}(\mathbf{E}^\top \mathbf{K} \mathbf{E}) - \alpha \log |\mathbf{I} + \beta^{-1} \mathbf{E}^\top \mathbf{E}|$ ▷ Equation (15)
 - 6: $\phi', \theta', \rho' \leftarrow \text{Update } \phi, \theta, \rho \text{ to minimize } l_{xy} + \lambda l_{\text{GMRF}}$
-

observed features in our case, and it makes an imbalance between nodes with and without observed features. Thus, we use the identity matrix $\mathbf{I} \in \mathbb{R}^{n \times n}$ as the input of f , using the observed features only as the answer for the training of SVGA. This allows f to learn an independent embedding for each node at its first layer and to have sufficient capacity to generate diverse node representations.

If we use a GCN with two layers as in previous work [18], the encoder function f is defined as $f(\mathbf{A}; \phi) = \hat{\mathbf{A}}(\sigma(\tilde{\mathbf{A}}\mathbf{W}_1))\mathbf{W}_2$, where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ is the normalized adjacency matrix, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self-loops, $\tilde{\mathbf{D}}$ is the degree matrix such that $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and σ is the ReLU function. $\mathbf{W}_1 \in \mathbb{R}^{n \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d \times d}$ are the weight matrices of layers 1 and 2, respectively, where n is the number of nodes, and d is the size of latent variables. We do not represent the bias terms for brevity. Note that the node feature matrix of the original formulation of GCN [18] is replaced with the identity matrix \mathbf{I} based on our idea of identity node features.

Unit normalization. A possible limitation of introducing the identity feature matrix is the large size of \mathbf{W}_1 , which can make the training process unstable. Thus, we project the latent representations \mathbf{Z} generated from the encoder f into a unit hypersphere by normalizing each vector of node i as $\mathbf{z}_i / \|\mathbf{z}_i\|_2$. This does not alter the main functionality of making diverse representations of nodes for making high-dimensional features, but improves the stability of training by restricting the output space [31].

3.1.2 Decoder Networks. We propose two decoder networks g_x and g_y to model $p_\theta(\mathbf{x}_i | \mathbf{z}_i)$ and $p_\rho(\mathbf{y}_i | \mathbf{z}_i)$, respectively. We assume that latent variables \mathbf{Z} have sufficient information to construct the observed features and labels. Thus, we minimize the complexity of decoder networks by adopting the simplest linear transformation as $g_x(\mathbf{z}_i) = \mathbf{W}_x \mathbf{z}_i + \mathbf{b}_x$ and $g_y(\mathbf{z}_i) = \mathbf{W}_y \mathbf{z}_i + \mathbf{b}_y$, where $\mathbf{W}_x \in \mathbb{R}^{m \times d}$, $\mathbf{W}_y \in \mathbb{R}^{c \times d}$, $\mathbf{b}_x \in \mathbb{R}^m$ and $\mathbf{b}_y \in \mathbb{R}^c$ are learnable weights and biases, m is the number of features, and c is the number of classes.

3.1.3 Optimization. We update the parameters of all the networks f , g_x , and g_y in an end-to-end way. We rewrite the ELBO of Equation (4) as the following objective function to be minimized:

$$l(\Theta) = \sum_{i \in \mathcal{V}_x} l_x(\hat{\mathbf{x}}_i, \mathbf{x}_i) + \sum_{i \in \mathcal{V}_y} l_y(\hat{\mathbf{y}}_i, \mathbf{y}_i) + \lambda l_{\text{GMRF}}(\mathbf{Z}, \mathbf{A}), \quad (6)$$

where l_x and l_y are loss terms for features and labels, respectively. l_{GMRF} is our proposed regularizer, whose details are described in Sections 3.2 and 3.3 through the process of structured inference. We use a hyperparameter λ for the amount of regularization.

The loss terms l_x and l_y are determined by how we model the distributions $p_\theta(\mathbf{x}_i | \mathbf{z}_i)$ and $p_\rho(\mathbf{y}_i | \mathbf{z}_i)$ following the distribution of true data. Common distributions for features include Gaussian, Bernoulli, and categorical (or one-hot) distributions:

$$l_x(\hat{\mathbf{x}}_i, \mathbf{x}_i) = \begin{cases} l_{\text{gau}}(\hat{\mathbf{x}}_i, \mathbf{x}_i) & \text{if } \mathbf{x}_i \text{ is continuous} \\ l_{\text{ber}}(\hat{\mathbf{x}}_i, \mathbf{x}_i) & \text{if } \mathbf{x}_i \text{ is binary} \\ l_{\text{cat}}(\hat{\mathbf{x}}_i, \mathbf{x}_i) & \text{if } \mathbf{x}_i \text{ is categorical,} \end{cases} \quad (7)$$

where the specific loss terms are defined as follows:

$$l_{\text{gau}}(\hat{\mathbf{x}}_i, \mathbf{x}_i) = -\sum_k (x_{ik} - \hat{x}_{ik})^2 \quad (8)$$

$$l_{\text{ber}}(\hat{\mathbf{x}}_i, \mathbf{x}_i) = -\sum_k (\alpha x_{ik} \log \sigma(\hat{x}_{ik}) + (1 - \alpha)(1 - x_{ik}) \log(1 - \sigma(\hat{x}_{ik}))) \quad (9)$$

$$l_{\text{cat}}(\hat{\mathbf{x}}_i, \mathbf{x}_i) = -\sum_k x_{ik} \log \text{softmax}(\hat{x}_{ik}). \quad (10)$$

$\hat{\mathbf{x}}_i = g_x(\mathbf{z}_i)$ is the output of the feature decoder, and σ is the logistic sigmoid function. We introduce α in Equation (9) to balance the effects of zero and nonzero entries of true features based on their occurrences [3]; α is the ratio of zero entries in the observed feature matrix. For the output $\mathbf{y}_i = g_y(\mathbf{z}_i)$ of the label decoder, we use the categorical loss, i.e., $l_y = l_{\text{cat}}$, due to the property of labels.

Algorithm 1 summarizes the training process of SVGA. It makes latent variables and predictions in lines 1 and 2, respectively, and computes the error between predictions and observations in line 3. Then, it computes our regularizer function in lines 4 and 5, whose information is described in the following subsections, to update the parameters of all three networks in an end-to-end way.

3.2 Structured Variational Inference

Previous works utilizing variational inference [16, 17] assume the prior of latent variables as a multivariate Gaussian distribution with identity covariance matrices, and run inference independently for each variable. This assumption is inappropriate in our case, since the correlations between variables, represented as a graph, are the main evidence in our graph-based learning.

We thus model the prior distribution $p(\mathbf{Z} | \mathbf{A})$ of Equation (4) as Gaussian Markov random field (GMRF) to incorporate the graph structure in the probabilistic modeling of variables. Specifically, we model $p(\mathbf{Z} | \mathbf{A})$ as GMRF $\mathcal{N}(\mathbf{0}, \mathbf{K}^{-1})$ with parameters $\mathbf{h} = \mathbf{0}$ and \mathbf{K} . We make the information matrix \mathbf{K} from \mathbf{A} as a graph Laplacian matrix with symmetric normalization [38]:

$$\mathbf{K} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}, \quad (11)$$

where \mathbf{I} is the identity matrix, and \mathbf{D} is the degree matrix such that $D_{ii} = \sum_j A_{ij}$. The resulting \mathbf{K} preserves the structural information of the graph G as a positive-semidefinite matrix that satisfies the constraint of GMRF; the nonzero entries of \mathbf{K} except the diagonal ones correspond to those of \mathbf{A} . Note that \mathbf{K} is a constant, since it represents the fixed prior distribution of variables.

We also model our target distribution $q_\phi(\mathbf{Z} | \mathbf{X}, \mathbf{y}, \mathbf{A})$ as a multivariate Gaussian distribution $\mathcal{N}(\mathbf{U}, \Sigma)$, where \mathbf{U} and Σ are the mean and covariance matrices of size $n \times d$ and $n \times n$, respectively. We assume that all d elements at each node share the same covariance matrix. \mathbf{U} and Σ are generated from encoder functions f_μ and f_σ , respectively, which contain the set ϕ of learnable parameters.

Given the Gaussian modelings of $q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$ and $p(\mathbf{Z} \mid \mathbf{A})$, the KL divergence is formulated as follows:

$$D_{\text{KL}}(q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A}) \parallel p(\mathbf{Z} \mid \mathbf{A})) = 0.5(\text{tr}(\mathbf{U}^\top \mathbf{K} \mathbf{U}) + d(\text{tr}(\mathbf{K} \Sigma) - \log |\Sigma|)) + C, \quad (12)$$

where C is a constant related to \mathbf{K} and $|\mathbf{V}|$. The goal of minimizing the KL divergence is to update ϕ of encoder functions to make q_ϕ similar to $p(\mathbf{Z} \mid \mathbf{A})$ as a regularizer of latent variables.

The computational bottleneck of Equation (12) is $\log |\Sigma|$, whose computation is $O(n^3)$ [10]. Thus, we decompose the covariance as $\Sigma = \beta \mathbf{I} + \mathbf{V} \mathbf{V}^\top$ with a rectangular matrix $\mathbf{V} \in \mathbb{R}^{n \times r}$, where β and r are hyperparameters such that $r \ll n$ [25]. As a result, $\log |\Sigma|$ is computed efficiently by the matrix determinant lemma [11]:

$$\log |\Sigma| = \log |\mathbf{I}_r + \beta^{-1} \mathbf{V}^\top \mathbf{V}| + \log |\beta \mathbf{I}_n|, \quad (13)$$

where \mathbf{I}_r and \mathbf{I}_n are the identity matrices of sizes $r \times r$ and $n \times n$, respectively. The computation of Equation (13) is $O(r^2 n + r^3)$, which is tractable even in graphs with a large number of nodes.

For each inference, we sample \mathbf{Z} randomly from q_ϕ based on \mathbf{U} and \mathbf{V} generated from f_μ and f_σ , respectively. Since the gradient-based update is not possible with the direct sampling of \mathbf{Z} , we use the reparametrization trick of variational autoencoders [16, 25]:

$$\mathbf{Z} = \mathbf{U} + \sqrt{\beta} \mathbf{M}_1 + \mathbf{V} \mathbf{M}_2, \quad (14)$$

where $\mathbf{M}_1 \in \mathbb{R}^{n \times d}$ and $\mathbf{M}_2 \in \mathbb{R}^{r \times d}$ are matrices of standard normal variables, which are sampled randomly at each time to simulate the sampling of \mathbf{Z} while supporting the backpropagation. The detailed process of inference is described in Appendix B.

We verify that the variables \mathbf{Z} sampled from Equation (14) follow the target distribution $\mathcal{N}(\mathbf{U}, \Sigma)$ by Lemmas 3.1 and 3.2.

LEMMA 3.1. *Let \mathbf{z}_i be a latent variable sampled from Equation (14) for node i , and \mathbf{u}_i be the i -th row of \mathbf{U} . Then, $\mathbb{E}[\mathbf{z}_i] = \mathbf{u}_i$.*

PROOF. See Appendix A.2. \square

LEMMA 3.2. *Assume that the size d of latent variables is one. Let \mathbf{z}_i and \mathbf{z}_j be latent variables sampled from Equation (14) for nodes i and j , respectively. Then, $\mathbb{E}[(\mathbf{z}_i - \mathbb{E}[\mathbf{z}_i])(\mathbf{z}_j - \mathbb{E}[\mathbf{z}_j])] = \Sigma_{ij}$.*

PROOF. See Appendix A.3. \square

3.3 Unified Deterministic Modeling

The reparametrization trick of variational inference requires us to sample different \mathbf{Z} at each inference to approximate the expectation term in Equation (5). However, this sampling process makes the training unstable, considering the characteristics of our feature estimation problem where 1) the inference is done for all nodes at once, not for each node independently, and 2) only a part of target variables have meaningful observations. Even a small perturbation for each node can result in a catastrophic change of the prediction, since we consider the correlations between nodes in $\mathcal{N}(\mathbf{U}, \Sigma)$.

We propose two ideas for improving the basic parameterization. First, we unify the parameter matrices \mathbf{U} and \mathbf{V} as a single matrix \mathbf{E} , and generate it from an encoder function f . This is based on the observation that \mathbf{U} and \mathbf{V} have similar roles of representing target nodes as low-dimensional vectors based on the graphical structure. Second, we change the stochastic sampling of \mathbf{Z} from $\mathcal{N}(\mathbf{E}, \Sigma)$ into a deterministic process that returns \mathbf{E} at every inference, which has

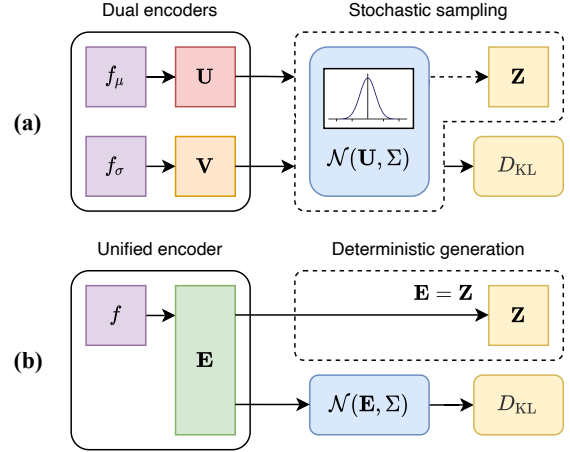


Figure 4: Comparison between the encoder structures of the (a) basic parameterization and (b) unified modeling. We use a single encoder f to deterministically generate \mathbf{Z} while utilizing the strong regularization of the KL divergence.

the largest probability in the distribution of \mathbf{Z} . This improves the stability of inference, while still allowing us to regularize the distribution of \mathbf{Z} with the KL divergence. Figure 4 depicts the difference between the basic parameterization and the unified modeling.

This unified modeling makes the KL divergence of Equation (12) into a general regularizer function that works with deterministic inference of node representations. First, we show in Lemma 3.3 that the first two terms of the right hand side of Equation (12) become equivalent as we assume $\mathbf{E} = \mathbf{U} = \mathbf{V}$ by the unified modeling.

LEMMA 3.3. *Let $\mathbf{K} \in \mathbb{R}^{n \times n}$, $\mathbf{E} \in \mathbb{R}^{n \times d}$, and $\Sigma = \beta \mathbf{I} + \mathbf{E} \mathbf{E}^\top$. Then, $\text{tr}(\mathbf{K} \Sigma) = \text{tr}(\mathbf{E}^\top \mathbf{K} \mathbf{E}) + C$, where C is a constant unrelated to \mathbf{E} .*

PROOF. See Appendix A.4. \square

Then, we propose our regularizer function used in Algorithm 1 by rewriting the KL divergence of Equation (12) as follows:

$$l_{\text{GMRF}}(\mathbf{E}, \mathbf{A}) = \text{tr}(\mathbf{E}^\top \mathbf{K} \mathbf{E}) - \alpha \log |\mathbf{I} + \beta^{-1} \mathbf{E}^\top \mathbf{E}|, \quad (15)$$

where $\alpha > 0$ is a hyperparameter that controls the effect of the log determinant term. We set $\alpha = 1/2$ is all of our experiments.

The first term of l_{GMRF} is called the graph Laplacian regularizer and has been widely used in graph learning [1, 21]. Its minimization makes adjacent nodes have similar representations in \mathbf{E} , and the symmetric normalization of \mathbf{K} alleviates the effect of node degrees in the regularization. The second term of l_{GMRF} can be considered as measuring the amount of space occupied by \mathbf{E} . In other words, its maximization makes $\mathbf{e}_1, \dots, \mathbf{e}_n$ distributed sparsely, alleviating the effect of $\text{tr}(\mathbf{E}^\top \mathbf{K} \mathbf{E})$ that squeezes the embeddings into a small space. The hyperparameter β controls the balance between the two terms having opposite goals; the second term is ignored if $\beta = \infty$, which means that the target nodes have no correlations.

3.4 Complexity Analysis

We analyze the time and space complexities of SVGA, assuming a graph convolutional network having two layers as the encoder function f . We define a space complexity as the amount of space

Table 1: Evaluation of SVGA and baseline approaches for missing feature estimation with respect to (top) recall and (bottom) nDCG. The best is in bold, and the second best is underlined. Our SVGA outperforms all baselines in most cases.

Metric	Model	Cora			Citeseer			Computers			Photo			Steam		
		@10	@20	@50	@10	@20	@50	@10	@20	@50	@10	@20	@50	@3	@5	@10
Recall	NeighAgg	.0906	.1413	.1961	.0511	.0908	.1501	.0321	.0593	.1306	.0329	.0616	.1361	.0603	.0881	.1446
	VAE	.0887	.1228	.2116	.0382	.0668	.1296	.0255	.0502	.1196	.0276	.0538	.1279	.0564	.0820	.1251
	GNN*	.1350	.1812	.2972	.0620	.1097	.2058	.0273	.0533	.1278	.0295	.0573	.1324	.2395	.3431	.4575
	GraphRNA	.1395	.2043	.3142	.0777	.1272	.2271	.0386	.0690	.1465	.0390	.0703	.1508	.2490	.3208	.4372
	ARWMF	.1291	.1813	.2960	.0552	.1015	.1952	.0280	.0544	.1289	.0294	.0568	.1327	.2104	.3201	.4512
	SAT	<u>.1653</u>	<u>.2345</u>	<u>.3612</u>	<u>.0811</u>	<u>.1349</u>	<u>.2431</u>	<u>.0421</u>	<u>.0746</u>	<u>.1577</u>	<u>.0427</u>	<u>.0765</u>	<u>.1635</u>	<u>.2536</u>	.3620	.4965
	SVGA	.1718	.2486	.3814	.0943	.1539	.2782	.0437	.0769	.1602	.0446	.0798	.1670	.2565	.3620	.4996
nDCG	NeighAgg	.1217	.1548	.1850	.0823	.1155	.1560	.0788	.1156	.1923	.0813	.1196	.1998	.0955	.1204	.1620
	VAE	.1224	.1452	.1924	.0601	.0839	.1251	.0632	.0970	.1721	.0675	.1031	.1830	.0902	.1133	.1437
	GNN*	.1791	.2099	.2711	.1026	.1423	.2049	.0673	.1028	.1830	.0712	.1083	.1896	.3366	.4138	.4912
	GraphRNA	.1934	.2362	.2938	.1291	.1703	.2358	.0931	.1333	.2155	.0959	.1377	.2232	.3437	.4023	.4755
	ARWMF	.1824	.2182	.2776	.0859	.1245	.1858	.0694	.1053	.1851	.0727	.1098	.1915	.3066	.3877	.4704
	SAT	<u>.2250</u>	<u>.2723</u>	<u>.3394</u>	<u>.1385</u>	<u>.1834</u>	<u>.2545</u>	<u>.1030</u>	<u>.1463</u>	<u>.2346</u>	<u>.1047</u>	<u>.1498</u>	<u>.2421</u>	.3585	.4400	<u>.5272</u>
	SVGA	.2381	.2894	.3601	.1579	.2076	.2892	.1068	.1509	.2397	.1084	.1549	.2472	<u>.3567</u>	<u>.4391</u>	.5299

Table 2: Summary of datasets.

Dataset	Type	Nodes	Edges	Feat.	Classes
Cora ¹	Binary	2,708	5,429	1,433	7
Citeseer ¹	Binary	3,327	4,732	3,703	6
Photo ²	Binary	7,650	119,081	745	8
Computers ²	Binary	13,752	245,861	767	10
Steam ³	Binary	9,944	266,981	352	1
Pubmed ¹	Continuous	19,717	44,324	500	3
Coauthor ²	Continuous	18,333	81,894	6,805	15
Arxiv ⁴	Continuous	169,343	1,157,799	128	40

¹ <https://github.com/kimiyoung/planetoid>

² <https://github.com/shchur/gnn-benchmark>

³ <https://github.com/xuChenSJTU/SAT-master-online>

⁴ <https://ogb.stanford.edu/docs/nodeprop/>

required to store intermediate data during each inference. Let d , m , and c be the size of latent variables, the number of node features, and the number of labels, respectively.

LEMMA 3.4. *Given a graph $G = (\mathcal{V}, \mathcal{E})$, the time complexity of SVGA is $O((d^2 + md + cd)|\mathcal{V}| + d|\mathcal{E}|)$ for each inference.*

PROOF. See Appendix A.5. \square

LEMMA 3.5. *Given a graph $G = (\mathcal{V}, \mathcal{E})$, the space complexity of SVGA is $O((d + m + c)|\mathcal{V}| + |\mathcal{E}| + d^2 + md + cd)$ for each inference.*

PROOF. See Appendix A.6. \square

Lemmas 3.4 and 3.5 show that SVGA is an efficient method whose complexity is linear with both the numbers of nodes and edges of the graph. The GMRF regularizer does not affect the inference of SVGA, because it is used only at the training time. Still, the time and space complexities of the GMRF loss l_{GMRF} of Equation (15) are $O(d^2|\mathcal{V}| + d|\mathcal{E}| + d^3)$ and $O(d|\mathcal{V}| + |\mathcal{E}| + d^2)$, respectively, which are linear with both the numbers of nodes and edges.

4 EXPERIMENTS

We perform experiments to answer the following questions:

Table 3: Evaluation for missing feature estimation on continuous features. The best is in bold, and the second best is underlined. RMSE is lower the better, while CORR is higher the better. “o.o.m.” refers to an out-of-memory error.

Model	Pubmed		Coauthor		Arxiv	
	RMSE	CORR	RMSE	CORR	RMSE	CORR
NeighAgg	0.0186	-0.2133	0.0952	-0.2279	0.1291	-0.4943
VAE	0.0170	-0.0236	0.0863	-0.0237	0.1091	-0.4773
GNN*	0.0168	-0.0010	0.0850	0.0179	0.1091	0.0283
GraphRNA	0.0172	-0.0352	0.0897	-0.1052	0.1131	-0.0419
ARWMF	<u>0.0165</u>	<u>0.0434</u>	0.0827	0.0710	o.o.m.	o.o.m.
SAT	<u>0.0165</u>	0.0378	<u>0.0820</u>	<u>0.0958</u>	<u>0.1055</u>	<u>0.0868</u>
SVGA	0.0158	0.1169	0.0798	0.1488	0.1005	0.1666

- Q1. **Feature estimation (Section 4.2).** Does SVGA show higher accuracy in feature estimation than those of baselines?
- Q2. **Node classification (Section 4.3).** Are the features generated by SVGA meaningful for node classification?
- Q3. **Effect of observed labels (Section 4.4).** Does the observation of labels help generating more accurate features?
- Q4. **Scalability (Section 4.5).** How does the computational time of SVGA increase with the number of edges?
- Q5. **Ablation study (Section 4.6).** How does the performance of SVGA for feature estimation change by the GMRF regularizer and the deterministic modeling of inference?

4.1 Experimental Setup

We introduce our experimental setup including datasets, baseline methods, evaluation metrics, and training processes.

Datasets. We use graph datasets summarized in Table 2, which were used in previous works [3, 20, 23, 30]. Node features in Cora, Citeseer, Photo, Computers, and Steam are zero-one binary vectors, while those in Pubmed, Coauthor, and ArXiv are continuous. Each node has a single discrete label. All nodes in Steam have the same class, and thus the dataset is not used for node classification.

Table 4: Comparison between SVGA and baselines by node classification accuracy, where each classifier is trained with the generated features. SVGA outperforms all baseline methods in most cases. “o.o.m.” refers to an out-of-memory error.

Model	Cora		Citeseer		Computers		Photo		Pubmed		Coauthor		Arxiv	
	MLP	GCN	MLP	GCN	MLP	GCN	MLP	GCN	MLP	GCN	MLP	GCN	MLP	GCN
NeighAgg	.6248	.8365	.5150	.6494	.8715	.6564	.5549	.8846	.7562	.5413	.9010	.8031	.3979	.6493
VAE	.2826	.3747	.4008	.3011	.4023	.4007	.2551	.2598	.2317	.2663	.3781	.2335	.1633	.1965
GNN*	.4852	.3747	.4013	.5779	.4034	.4203	.3933	.2598	.2317	.4278	.3789	.2335	.2607	.4721
GraphRNA	.7581	.6968	.6035	.8198	.8650	.8172	.6320	.8407	.7710	.6394	.9207	.8851	.1609	.1859
ARWMF	.7769	.5608	.6180	.8205	.7400	.8089	.2267	.4675	.2320	.2764	.6146	.8347	o.o.m.	o.o.m.
SAT	.7937	.8201	.4618	.8579	.8766	.7439	.6475	.8976	.7672	.6767	.9260	.8402	.3144	.5677
SVGA (proposed)	.8493	.8806	.6227	.8533	.8854	.8808	.6757	.9209	.8293	.6879	.9264	.9037	.4394	.6644

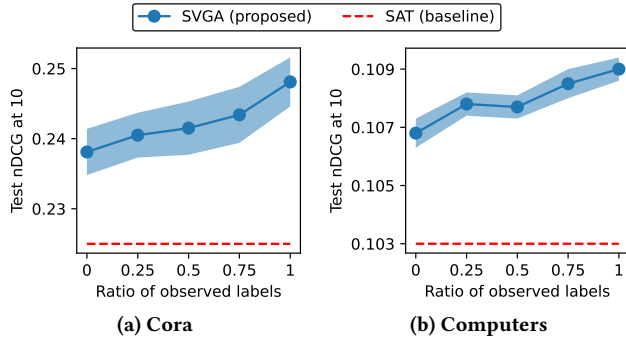


Figure 5: The accuracy of SVGA for feature estimation with additional observations of labels. We show the average and standard deviation of ten runs. SVGA effectively utilizes the given labels, making more accurate predictions.

Baselines. We compare SVGA with existing models for feature estimation. NeighAgg [24] is a simple approach that aggregates the features of neighboring nodes through mean pooling. VAE [16] is a generative model that learns latent representations of examples. GCN [18], GraphSAGE [9], and GAT [26] are popular graph neural networks that have been used in various domains. We report the best performance among the three models as GNN* for brevity.

GraphRNA [13] and ARWMF [2] are recent methods for representation learning, which can be applied for generating features. SAT [3] is the state-of-the-art model for missing feature estimation, which trains separate autoencoders with a shared latent space for the features and graphical structure, respectively. We use GAT and GCN as the backbone network of SAT in datasets with discrete and continuous features, respectively, which are the settings that show the best performance in the original paper [3].

Evaluation metrics. We evaluate the performance of feature estimation with four evaluation metrics. For binary features, we treat each nonzero entry as a target item, considering the task as a ranking problem to find all nonzero entries. Recall at k measures the ratio of true entries contained in the top k predictions for each node, while nDCG at k measures the overall quality of predicted scores in terms of information retrieval. We vary k over $\{3, 5, 10\}$ in the Steam dataset and $\{10, 20, 50\}$ in the other datasets, because Steam has fewer features and thus a prediction is generally easier. For continuous features, we compare the predictions and the true features in an elementwise way with the root mean squared error

(RMSE) and the square of the correlation coefficient (CORR). The definitions of evaluation metrics are in Appendix C.

Experimental process. We take different processes of experiments for feature estimation and node classification. For feature estimation, we split all nodes at each dataset into the training, validation, and test sets by the 4:1:5 ratio as in previous work [3]. We train each model based on the observed features of training nodes and find the parameters that maximize the validation performance. We run each experiment ten times and report the average.

For node classification, we take only the test nodes of feature estimation, whose features are generated by our SVGA or baseline models. Then, we perform the 5-fold cross-validation in the target nodes, evaluating the quality of generated features with respect to the accuracy of node classification. We use a multilayer perceptron (MLP) and GCN as classifiers. For the training and evaluation of GCN, we use the induced subgraph of target nodes.

Even though our SVGA can utilize observed labels as additional evidence, we do not assume the observation of labels unless otherwise noted. This is to make a fair comparison between SVGA and baseline models that assume only the observation of features. We perform experiments in Section 4.4 with observed labels.

Hyperparameters. The hyperparameter setting of our SVGA is described in Appendix D. For baselines, we take the experimental results from a previous work [3] that optimized the hyperparameters for the feature estimation problem on our datasets.

4.2 Performance on Feature Estimation (Q1)

Table 1 compares SVGA and baseline models for feature estimation. SVGA outperforms all baselines with a significant margin in most cases; SVGA shows up to 16.3% and 14.0% higher recall and nDCG, respectively, compared with the best competitors. The amount of improvement over baselines is the largest in Cora and Citeseer, which are similar citation graphs, and the smallest in Steam. This is because the citation graphs have high-dimensional features with sparse graph structures, increasing the difficulty of estimation for the baseline methods. On the other hand, Steam has the smallest number of features, while having the densest structure.

Table 3 presents the result of feature estimation for continuous features. SVGA still outperforms all baselines, and the amount of improvement is similar in all three datasets. The combination of Tables 1 and 3 shows that SVGA works well with various types of node features, providing stable performance. ARWMF causes an out-of-memory error in 256GB memory, due to the computation of A^n of the adjacency matrix A with large $n \geq 5$.

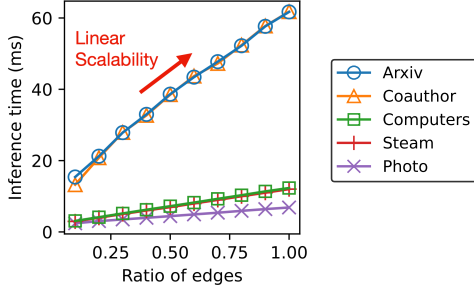


Figure 6: The inference time of SVGA in graphs of different sizes. We randomly sample nine subgraphs for each dataset. SVGA shows the linear scalability in all datasets.

4.3 Performance on Node Classification (Q2)

Table 4 shows the accuracy of node classification with two types of classifiers: MLP and GCN. MLP relies on the generated features for prediction, while GCN utilizes also the graph structure. SVGA outperforms all baselines in most cases, making a consistency with the results of feature estimation in Tables 1 and 3; SVGA achieves up to 10.4% and 7.8% higher accuracy in MLP and GCN, respectively, compared to the best competitors. The Steam dataset is excluded from Table 4, since it has the same label for all nodes.

4.4 Effect of Observed Labels (Q3)

Figure 5 shows the performance of SVGA for feature estimation with different ratios of observed labels. For instance, if the ratio is 0.5, half of all nodes have observed labels: $|\mathcal{V}_y| = 0.5|\mathcal{V}|$. Note that the experiments for Tables 1, 3, and 4 are done with no labels for a fair comparison with the baseline models; the results of these experiments correspond to the leftmost points in Figure 5. We also report the performance of SAT for comparison.

SVGA shows higher accuracy with more observations of labels in both datasets, demonstrating its ability to use labels to improve the performance of feature estimation. Since the parameters need to be optimized to predict both features and labels accurately, the observed labels work as an additional regularizer that guides the training of latent variables to avoid the overfitting.

4.5 Scalability (Q4)

Figure 6 shows the scalability of SVGA with respect to the number of edges on the five largest datasets in Table 2. For each dataset, we sample nine random subgraphs of different sizes from $0.1|\mathcal{E}|$ to $0.9|\mathcal{E}|$, where $|\mathcal{E}|$ denotes the number of original edges. We measure the inference time of SVGA in each graph ten times and report the average. The figure shows the linear scalability of SVGA with the number of edges in all datasets, supporting Lemma 3.4. Arxiv and Coauthor take the longest inference times, as Arxiv and Coauthor have the largest numbers of edges and features, respectively.

4.6 Ablation Study (Q5)

Figure 7 shows an ablation study that compares SVGA with its two variants SVGA-U and SVGA-R. SVGA-U runs stochastic inference described in Section 3.2, without our idea of unified deterministic modeling. The detailed process of stochastic inference is described

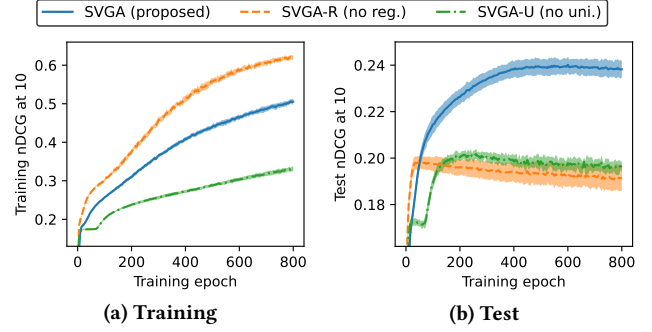


Figure 7: An ablation study of SVGA on Cora compared with its variants SVGA-R and SVGA-U (details in Section 4.6). We show the average and standard deviation of ten runs. SVGA makes the best test accuracy based on our proposed ideas.

also in Algorithm 2 of Appendix B. SVGA-R runs the deterministic inference but removes the regularizer term l_{GMRF} of Equation (15); it follows Algorithm 1 as in SVGA except for lines 4 and 5.

SVGA shows the best test accuracy during the training with a stable curve. The training accuracy is the best with SVGA-R, since it overfits to training nodes without the regularizer term. On the other hand, the training accuracy of SVGA-U is the lowest among the three methods, while its test accuracy becomes similar to that of SVGA-R at the later epochs. This is because SVGA-U fails even at maximizing the training accuracy due to the unstable training. The standard deviation of training accuracy is very small with all three methods, despite their different modelings.

5 RELATED WORKS

Graph neural networks. Graph neural networks (GNN) refer to deep neural networks designed for graph-structured data [9, 18, 26, 27, 36, 37]. Since GNNs require the features of all nodes, one needs to generate artificial features to apply a GNN to a graph with missing features. Derr et al. [5] and Cui et al. [4] generate features from the graph structure. Kipf and Welling [18] model the missing features as one-hot vectors, while Zhao and Akoglu [39] leave them as zero vectors and propose a new regularizer function.

Our SVGA enables a GNN to be applied to graphs with partial observations by estimating missing features. The main advantage of feature estimation is that the modification of a GNN classifier is not required, regardless of the number of observations given in the original graph. Previous works that directly deal with partially observed graphs require finding new hyperparameters [39] or even making a new weight matrix [18] when the number of observations changes, making it difficult to reuse a trained model.

Missing feature estimation. There are recent works that can be used directly for our feature estimation problem [2, 3, 13]. Such methods are adopted as the main competitors in our experiments. The main advantage of SVGA over the previous approaches is the strong regularizer that allows us to effectively propagate the partial observations to the entire graph, avoiding the overfitting problem even with large representation power for feature estimation.

GRAPE [37] estimates missing features in tabular data by learning a graph between examples and features. The main difference

from our work is that GRAPE assumes partial observations of feature elements, not feature vectors. In other words, GRAPE cannot be used to estimate the features of nodes that have no partial observations, which is the scenario assumed in our experiments.

Node representation learning. Unsupervised node representation learning [8] is to represent each node as a low-dimensional vector that summarizes its properties embedded in the structure and node features [7, 8, 22, 27, 28]. Such methods make embeddings in a latent space, while we aim to learn the representations of nodes in a high-dimensional feature space; the node features generated from our SVGA are interpretable in the feature domain.

Probabilistic modeling of graphs. Previous works model real-world graphs as pairwise Markov random fields with discrete variables and run graphical inference for node classification [14, 32–35]. Our work can be considered as a generalization of such works into the challenging task of missing feature estimation, which requires us to estimate high-dimensional continuous variables.

6 CONCLUSION

We propose SVGA (Structured Variational Graph Autoencoder), an accurate method for missing feature estimation. SVGA estimates high-dimensional features of nodes from a graph with partial observations, and its framework is carefully designed to model the target distributions of structured variational inference. The main idea of SVGA is the structural regularizer that assumes the prior of latent variables as Gaussian Markov random field, which considers the graph structure as the main evidence for modeling the correlations between nodes in variational inference. SVGA outperforms previous methods for feature estimation and node classification, achieving the state-of-the-art accuracy in benchmark datasets. Future works include extending the domain of SVGA into directed or heterogeneous graphs that are common in real-world datasets.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) [No.2020-0-00894, Flexible and Efficient Model Compression Method for Various Applications and Environments], [No.2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University)], and [NO.2021-0-0268, Artificial Intelligence Innovation Hub (Artificial Intelligence Institute, Seoul National University)]. The Institute of Engineering Research at Seoul National University provided research facilities for this work. The ICT at Seoul National University provides research facilities for this study. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1C1C1008526). U Kang is the corresponding author.

REFERENCES

- [1] Rie Kubota Ando and Tong Zhang. 2006. Learning on Graph with Laplacian Regularization. In *NIPS*.
- [2] Lei Chen, Shunwang Gong, Joan Bruna, and Michael Bronstein. 2019. Attributed random walk as matrix factorization. In *NeurIPS Workshop*.
- [3] Xu Chen, Siheng Chen, Jiangchao Yao, Huangjie Zheng, Ya Zhang, and Ivor W. Tsang. 2020. Learning on Attribute-Missing Graphs. *CoRR* abs/2011.01623 (2020). arXiv:2011.01623
- [4] Hejie Cui, Zijie Lu, Pan Li, and Carl Yang. 2021. On Positional and Structural Node Features for Graph Neural Networks on Non-attributed Graphs. *CoRR* abs/2107.01495 (2021). arXiv:2107.01495
- [5] Tyler Derr, Yao Ma, and Jiliang Tang. 2018. Signed Graph Convolutional Networks. In *ICDM*.
- [6] Chi Thang Duong, Thanh Dat Hoang, Ha The Hien Dang, Quoc Viet Hung Nguyen, and Karl Aberer. 2019. On Node Features for Graph Neural Networks. *CoRR* abs/1911.08795 (2019). arXiv:1911.08795
- [7] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*.
- [8] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40, 3 (2017), 52–74.
- [9] William L. Hamilton, Zhitaoying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.
- [10] Insu Han, Dmitry Malioutov, and Jinwoo Shin. 2015. Large-scale log-determinant computation through stochastic Chebyshev expansions. In *ICML*.
- [11] David A Harville. 1998. Matrix algebra from a statistician’s perspective.
- [12] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *NeurIPS*.
- [13] Xiao Huang, Qingquan Song, Yuening Li, and Xia Hu. 2019. Graph Recurrent Networks With Attributed Random Walks. In *KDD*.
- [14] Saehan Jo, Jaemin Yoo, and U. Kang. 2018. Fast and Scalable Distributed Loopy Belief Propagation on Real-World Graphs. In *WSDM*.
- [15] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
- [16] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *ICLR*.
- [17] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *CoRR* abs/1611.07308 (2016). arXiv:1611.07308
- [18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
- [19] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press.
- [20] Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-Based Recommendations on Styles and Substitutes. In *SIGIR*.
- [21] Jiahao Pang and Gene Cheung. 2017. Graph Laplacian Regularization for Image Denoising: Analysis in the Continuous Domain. *IEEE Trans. Image Process.* 26, 4 (2017), 1770–1785.
- [22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *KDD*.
- [23] Oleksandr Shchur, Maximilian Mummé, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *CoRR* abs/1811.05868 (2018). arXiv:1811.05868
- [24] Özgür Simsek and David D. Jensen. 2008. Navigating networks by using homophily and degree. *Proc. Natl. Acad. Sci. USA* 105, 35 (2008), 12758–12762.
- [25] Marcin Tomczak, Siddharth Swaroop, and Richard E. Turner. 2020. Efficient Low Rank Gaussian Variational Inference for Neural Networks. In *NeurIPS*.
- [26] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
- [27] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR*.
- [28] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *KDD*.
- [29] Yulei Yang and Dongsheng Li. 2020. NENN: Incorporate Node and Edge Features in Graph Neural Networks. In *ACML*.
- [30] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *ICML*.
- [31] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*.
- [32] Jaemin Yoo, Hyunsik Jeon, and U Kang. 2019. Belief Propagation Network for Hard Inductive Semi-Supervised Learning. In *IJCAI*.
- [33] Jaemin Yoo, Saehan Jo, and U Kang. 2017. Supervised Belief Propagation: Scalable Supervised Inference on Attributed Networks. In *ICDM*.
- [34] Jaemin Yoo, U Kang, Mauro Scanagatta, Giorgio Corani, and Marco Zaffalon. 2020. Sampling Subgraphs with Guaranteed Treewidth for Accurate and Efficient Graphical Inference. In *WSDM*.
- [35] Jaemin Yoo, Junghun Kim, Hoyoung Yoon, Geonsoo Kim, Changwon Jang, and U Kang. 2021. Accurate Graph-Based PU Learning without Class Prior. In *ICDM*.
- [36] Jiaxuan You, Jonathan M. Gomes-Selman, Rex Ying, and Jure Leskovec. 2021. Identity-aware Graph Neural Networks. In *AAAI*.
- [37] Jiaxuan You, Xiaobai Ma, Daisy Yi Ding, Mykel J. Kochenderfer, and Jure Leskovec. 2020. Handling Missing Data with Graph Representation Learning. In *NeurIPS*.
- [38] Cha Zhang, Dinei Florêncio, and Philip A Chou. 2015. Graph signal processing-a probabilistic framework. *Microsoft Res., Redmond, WA, USA, Tech. Rep. MSR-TR-2015-31* (2015).
- [39] Lingxiao Zhao and Leman Akoglu. 2020. PairNorm: Tackling Oversmoothing in GNNs. In *ICLR*.

A PROOFS OF LEMMAS

A.1 Proof of Lemma 2.1

PROOF. The probability density function of $\mathcal{N}(\mu, \Sigma)$ is

$$f(\mathbf{z}) = C' \exp(-(\mathbf{z} - \mu)^\top \Sigma^{-1} (\mathbf{z} - \mu)),$$

where $C' = (2\pi)^{-d/2} |\Sigma|^{-1/2}$ is a constant.

We rewrite f as follows with $\mathbf{K} = \Sigma^{-1}$ and $\mathbf{h} = \mathbf{K}\mu$:

$$\begin{aligned} f(\mathbf{z}) &= C' \exp(-\mathbf{z}^\top \mathbf{K} \mathbf{z} + 2\mu^\top \mathbf{K} \mathbf{z} - \mu^\top \mathbf{K} \mu) \\ &= C'' \exp(-\mathbf{z}^\top \mathbf{K} \mathbf{z} + 2\mu^\top \mathbf{K} \mathbf{z}) \\ &= C'' \exp\left(-\sum_i \sum_j z_i K_{ij} z_j + 2 \sum_i h_i z_i\right). \end{aligned}$$

where $C'' = \exp(\mu^\top \mathbf{K} \mu)$. C' is also a constant.

By the definition of GMRF, $K_{ij} \neq 0$ only if edge (i, j) exists in the given graph $G = (\mathcal{V}, \mathcal{E})$. Then, we rewrite $f(\mathbf{z})$ as

$$f(\mathbf{z}) = C \exp\left(\sum_{(i,j) \in \mathcal{E}} (-z_i K_{ij} z_j) + \sum_{i \in \mathcal{V}} (-0.5 K_{ii} z_i^2 + h_i z_i)\right),$$

where $C = \exp(2) \cdot C''$. We prove the lemma by substituting $\psi_{ij}(z_{ij})$ and $\psi_i(z_i)$ for the first and second terms, respectively. \square

A.2 Proof of Lemma 3.1

PROOF. The random variables included in Equation (14) are \mathbf{M}_1 and \mathbf{M}_2 . Since \mathbf{M}_1 and \mathbf{M}_2 are filled with standard normal values, it is satisfied that $\mathbb{E}[\sqrt{\beta} \mathbf{M}_1] = \mathbf{0}$ and $\mathbb{E}[\mathbf{V} \mathbf{M}_2] = \mathbf{0}$, regardless of the actual values of β and \mathbf{V} . Thus, $\mathbb{E}(\mathbf{Z}) = \mathbb{E}(\mathbf{U}) = \mathbf{U}$. \square

A.3 Proof of Lemma 3.2

PROOF. The following is satisfied for both $k = i$ and $k = j$ based on Lemma 3.1:

$$z_k - \mathbb{E}[z_k] = \sqrt{\beta} \mathbf{m}_{1k} + \mathbf{v}_k^\top \mathbf{m}_2,$$

where \mathbf{v}_k and \mathbf{m}_2 are r -dimensional vectors.

Then, the covariance between z_i and z_j is given as

$$\begin{aligned} \mathbb{E}[(z_i - \mathbb{E}[z_i])(z_j - \mathbb{E}[z_j])] &= \beta \mathbb{E}[m_{1i} m_{1j}] \\ &+ \sqrt{\beta} \mathbf{v}_i^\top \mathbb{E}[m_{1i} \mathbf{m}_2] + \sqrt{\beta} \mathbf{v}_j^\top \mathbb{E}[m_{1j} \mathbf{m}_2] + \mathbb{E}[\mathbf{v}_i^\top \mathbf{m}_2 \mathbf{v}_j^\top \mathbf{m}_2]. \end{aligned}$$

Since every element of \mathbf{M}_1 and \mathbf{M}_2 follows the standard normal distribution, the following are satisfied. First, $\mathbb{E}[m_{1i} m_{1j}] = 1$ if $i = j$ and zero otherwise. Second, the second and third elements of the right hand side are zero. Third, $\mathbb{E}[\mathbf{v}_i^\top \mathbf{m}_2 \mathbf{v}_j^\top \mathbf{m}_2] = \mathbf{v}_i^\top \mathbf{v}_j$. As a result, we have the following equality:

$$\mathbb{E}[(z_i - \mathbb{E}[z_i])(z_j - \mathbb{E}[z_j])] = \beta \mathbb{I}[i = j] + \mathbf{v}_i^\top \mathbf{v}_j,$$

where \mathbb{I} is an indicator function that returns one if the condition holds, and zero otherwise. This equation is the same as the definition of $\Sigma = \beta \mathbf{I} + \mathbf{V} \mathbf{V}^\top$ in the matrix form. \square

A.4 Proof of Lemma 3.3

PROOF. $\text{tr}(\mathbf{K} \Sigma) = \beta \text{tr}(\mathbf{K}) + \text{tr}(\mathbf{K} \mathbf{E} \mathbf{E}^\top)$ due to the definition of Σ . The cyclic property of a trace makes $\text{tr}(\mathbf{K} \mathbf{E} \mathbf{E}^\top) = \text{tr}(\mathbf{E}^\top \mathbf{K} \mathbf{E})$. Thus, $\text{tr}(\mathbf{K} \Sigma) = \beta \text{tr}(\mathbf{K}) + \text{tr}(\mathbf{E}^\top \mathbf{K} \mathbf{E})$, and $\text{tr}(\mathbf{K})$ is a constant. \square

Algorithm 2 Basic version of structured variational inference.

Input: Adjacency matrix \mathbf{A} , diagonal adjacency \mathbf{D} , feature \mathbf{X} , (optional) one-hot label \mathbf{Y} , hyperparameter β , networks f_μ, f_σ, g_x , and g_y , and their parameters $\phi_\mu, \phi_\sigma, \theta$, and ρ , resp.

Output: Updated parameters $\phi'_\mu, \phi'_\sigma, \theta'$, and ρ'

- 1: $\mathbf{U}, \mathbf{V} \leftarrow f_\mu(\mathbf{A}; \phi_\mu), f_\sigma(\mathbf{A}; \phi_\sigma)$ \triangleright Run encoder functions
 - 2: $\Sigma \leftarrow \beta \mathbf{I} + \mathbf{V} \mathbf{V}^\top$ \triangleright Make the covariance matrix
 - 3: $\mathbf{M}_1, \mathbf{M}_2 \leftarrow \text{StandardNormal}()$ \triangleright Sample random matrices
 - 4: $\mathbf{Z} \leftarrow \mathbf{U} + \sqrt{\beta} \mathbf{M}_1 + \mathbf{V} \mathbf{M}_2$ \triangleright Make latent variables
 - 5: $\hat{\mathbf{X}}, \hat{\mathbf{Y}} \leftarrow g_x(\mathbf{Z}, \mathbf{A}; \theta), g_y(\mathbf{Z}, \mathbf{A}; \rho)$ \triangleright Make predictions
 - 6: $l_{xy} \leftarrow \sum_i l_x(\hat{\mathbf{x}}_i, \mathbf{x}_i) + \sum_j l_y(\hat{\mathbf{y}}_j, \mathbf{y}_j)$ \triangleright Equation (7) to (10)
 - 7: $\mathbf{K} \leftarrow \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$ \triangleright Equation (11)
 - 8: $l_{\text{KLD}} \leftarrow 0.5(\text{tr}(\mathbf{U}^\top \mathbf{K} \mathbf{U}) + d(\text{tr}(\mathbf{K} \Sigma) - \log |\Sigma|))$ \triangleright Equation (12)
 - 9: $\phi'_\mu, \phi'_\sigma, \theta', \rho' \leftarrow \text{Update } \phi_\mu, \phi_\sigma, \theta, \rho \text{ to minimize } l_{xy} + l_{\text{KLD}}$
-

A.5 Proof of Lemma 3.4

PROOF. SVGA consists of an encoder f and two decoders g_x and g_y . The complexity of f is $O(d^2|\mathcal{V}| + d|\mathcal{E}|)$ assuming the identity feature matrix. The complexities of g_x and g_y are $O(md|\mathcal{V}|)$ and $O(cd|\mathcal{V}|)$, respectively. \square

A.6 Proof of Lemma 3.5

PROOF. SVGA consists of an encoder f and two decoders g_x and g_y . The complexity of f is $O(d|\mathcal{V}| + |\mathcal{E}| + d^2)$ assuming the identity feature matrix. The complexities of g_x and g_y are $O(m|\mathcal{V}| + d|\mathcal{V}| + md)$ and $O(c|\mathcal{V}| + d|\mathcal{V}| + cd)$, respectively. \square

B DETAILS OF STOCHASTIC INFERENCE

We present a detailed algorithm of structured variational inference in Algorithm 2, which performs stochastic sampling described in Section 3.2. It uses two encoder functions for generating embedding matrices \mathbf{U} and \mathbf{V} , respectively, in line 1. Then, it samples \mathbf{Z} from the Gaussian distribution with the reparametrization trick in lines 2 to 4. The prediction is done as in the deterministic inference, but the regularizer term works differently in line 8, taking \mathbf{U} and Σ as its inputs. The parameters of all four networks are updated.

C EVALUATION METRICS

We use four metrics for the evaluation of estimated features: recall at k and nDCG at k for binary features, and RMSE and CORR for continuous features. Categorical features are not included in our datasets in Table 2, but we can use classification accuracy as done for evaluating labels. The symbols used in this section are defined as follows: n is the number of nodes, d is the number of features, \mathbf{x}_i is the true feature vector of node i , $\hat{\mathbf{x}}_i$ is the prediction for \mathbf{x}_i , x_{ij} is the j -th element of \mathbf{x}_i , and $\mathbb{I}(\cdot)$ is a binary function that returns one if the condition holds and zero otherwise.

Evaluation of binary features. We consider the feature estimation for binary features as a ranking problem, which is to find the nonzero elements at each feature vector \mathbf{x}_i of node i . Let \hat{r}_{ij} be the index having the j -th largest score in $\hat{\mathbf{x}}_i$. Then, we use the top k predictions with the largest scores, i.e., $\{\hat{x}_{i\ell} \mid \ell = \hat{r}_{i1}, \dots, \hat{r}_{ik}\}$, at each node i , where k is chosen in $\{3, 5, 10, 20, 50\}$. This is done also in previous work for binary feature estimation [3] to focus more on the predictive performance of the top predictions.

Recall at k measures the ratio of true entries contained in the top k predictions for each node:

$$\text{REC}_k(\hat{\mathbf{X}}, \mathbf{X}) = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k \frac{\mathbb{I}[x_i, \hat{r}_{ij} = 1]}{\|\mathbf{x}_i\|_0}, \quad (16)$$

where $\|\mathbf{x}_i\|_0$ is the number of nonzero entries in \mathbf{x}_i . For instance, recall @ 3 is computed as 2/3 in the following example:

$$\mathbf{x}_i = (0, 0, 1, 1, 1)$$

$$\hat{\mathbf{x}}_i = (0.1, 0.7, 0.2, 0.8, 0.9),$$

since $\hat{r}_{i1} = 5$, $\hat{r}_{i2} = 4$, and $\hat{r}_{i3} = 2$, and two of the nonzero entries of \mathbf{x}_i are included in the top 3 predictions with the largest scores.

nDCG at k measures also the quality of order in the top k predictions with respect to information retrieval. nDCG is computed by normalizing DCG at k , which is defined as

$$\text{DCG}_k(\hat{\mathbf{X}}, \mathbf{X}) = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k \frac{\mathbb{I}(x_i, \hat{r}_{ij} = 1)}{\log_2(j+1)}. \quad (17)$$

Evaluation of continuous features. We evaluate predictions for continuous features with simple metrics of RMSE and CORR. RMSE measures an error between predictions and true features:

$$\text{RMSE}(\hat{\mathbf{X}}, \mathbf{X}) = \frac{1}{n} \sum_{i=1}^n \sqrt{\frac{1}{d} \sum_{j=1}^d (\hat{x}_{ij} - x_{ij})^2}. \quad (18)$$

CORR measures how much predictions and true features are correlated, and is defined as follows:

$$\text{CORR}(\hat{\mathbf{X}}, \mathbf{X}) = \frac{1}{d} \sum_{j=1}^d \left(1 - \frac{\sum_{i=1}^n (\hat{x}_{ij} - x_{ij})^2}{\sum_{i=1}^n (x_{ij} - \bar{x}_j)^2} \right), \quad (19)$$

where $\bar{x}_j = \sum_{i=1}^n x_{ij}/n$ is the mean of the j -th feature of all nodes. CORR is higher the better, while RMSE is lower the better.

D HYPERPARAMETER SETTINGS

We search the hyperparameters of our SVGA as follows: the size d of latent variables in $\{256, 512\}$, the dropout probability in $\{0.0, 0.5\}$, the regularization parameters λ and β in $\{0.01, 0.1, 1.0\}$, and the unit normalization of latent variables in $\{\text{true}, \text{false}\}$. We also use the Adam [15] optimizer with the learning rate $r = 0.005$ in Steam and $r = 0.001$ in all other datasets. The early stopping is used with the validation performance, and all of our experiments were done at a workstation with RTX 2080 based on PyTorch. More detailed information can be found in our official implementation.¹

¹<https://github.com/snudatalab/SVGA>