Bayesian Graph Neural Networks with Adaptive Connection Sampling

Arman Hasanzadeh [*1] **Ehsan Hajiramezanali** [*1] **Shahin Boluki** [1] **Mingyuan Zhou** [2] **Nick Duffield** [1]
Krishna Narayanan [1] **Xiaoning Qian** [1]

Abstract

We propose a unified framework for adaptive connection sampling in graph neural networks (GNNs) that generalizes existing stochastic regularization methods for training GNNs. The proposed framework not only alleviates over-smoothing and over-fitting tendencies of deep GNNs, but also enables learning with uncertainty in graph analytic tasks with GNNs. Instead of using fixed sampling rates or hand-tuning them as model hyperparameters as in existing stochastic regularization methods, our adaptive connection sampling can be trained jointly with GNN model parameters in both global and local fashions. GNN training with adaptive connection sampling is shown to be mathematically equivalent to an efficient approximation of training Bayesian GNNs. Experimental results with ablation studies on benchmark datasets validate that adaptively learning the sampling rate given graph training data is the key to boosting the performance of GNNs in semi-supervised node classification, making them less prone to over-smoothing and over-fitting with more robust prediction.

1. Introduction

Graph neural networks (GNNs), and its numerous variants, have shown to be successful in graph representation learning by extracting high-level features for nodes from their topological neighborhoods. GNNs have boosted the state-of-the-art performance in a variety of graph analytic tasks, such as semi-supervised node classification and link prediction (Kipf & Welling, 2017; 2016; Hasanzadeh et al., 2019; Hajiramezanali et al., 2019). Despite their successes, GNNs have two major limitations: 1) they cannot go very deep due to *over-smoothing* and *over-fitting* phenomena (Li et al., 2018; Kipf & Welling, 2017); 2) the current implementations of GNNs do not provide uncertainty quantification (UQ) of output predictions.

There exist a variety of methods to address these problems. For example, DropOut (Srivastava et al., 2014) is a popular regularisation technique with deep neural networks (DNNs) to avoid over-fitting, where network units are randomly masked during training. In GNNs, DropOut is realized by randomly removing the node features during training (Rong et al., 2019). Often, the procedure is independent of the graph topology. However, empirical results have shown that, due to the nature of Laplacian smoothing in GNNs, graph convolutions have the over-smoothing tendency of mixing representations of adjacent nodes so that, when increasing the number of GNN layers, all nodes' representations will converge to a stationary point, making them unrelated to node features (Li et al., 2018). While it has been shown in Kipf & Welling (2017) that DropOut alone is ineffectual in preventing over-fitting, partially due to over-smoothing, the combination of DropEdge, in which a set of edges are randomly removed from the graph, with DropOut has recently shown potential to alleviate these problems (Rong et al., 2019).

On the other hand, with the development of efficient posterior computation algorithms, there have been successes in learning with uncertainty by Bayesian extensions of traditional deep network architectures, including convolutional neural networks (CNNs). However, for GNNs, deriving their Bayesian extensions is more challenging due to their irregular neighborhood connection structures. In order to account for uncertainty in GNNs, Zhang et al. (2019) present a Bayesian framework where the observed graph is viewed as a realization from a parametric family of random graphs. This allows joint inference of the graph and the GNN weights, leading to resilience to noise or adversarial attacks. Besides its prohibitive computational cost, the choice of the random graph model is important and can be inconsistent for different problems and datasets. Furthermore, the posterior inference in the current implementation only depends on the graph topology, but cannot consider node features.

[*]Equal contribution [1]Electrical and Computer Engineering Department, Texas A&M University, College Station, Texas, USA [2]McCombs School of Business, The University of Texas at Austin, Austin, Texas, USA. Correspondence to: Arman Hasanzadeh <armanihm@tamu.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

In this paper, we introduce a general stochastic regularization technique for GNNs by adaptive connection sampling—Graph DropConnect (GDC). We show that existing GNN regularization techniques such as DropOut (Srivastava et al., 2014), DropEdge (Rong et al., 2019), and node sampling (Chen et al., 2018) are special cases of GDC. GDC regularizes neighborhood aggregation in GNNs at each channel, separately. This prevents connected nodes in graph from having the same learned representations in GNN layers; hence better improvement without serious over-smoothing can be achieved. Furthermore, adaptively learning the connection sampling or drop rate in GDC enables better stochastic regularization given graph data for target graph analytic tasks. In fact, our ablation studies show that only learning the DropEdge rate, without any DropOut, already substantially improves the performance in semi-supervised node classification with GNNs. By probabilistic modeling of the *connection* drop rate, we propose a hierarchical beta-Bernoulli construction for Bayesian learnable GDC, and derive the solution with both continuous relaxation and direct optimization with Augment-REINFORCE-Merge (ARM) gradient estimates. With the naturally enabled UQ and regularization capability, our learnable GDC can help address both over-smoothing and UQ challenges to further push the frontier of GNN research.

We further prove that adaptive connection sampling of GDC at each channel can be considered as random aggregation and diffusion in GNNs, with a similar Bayesian approximation interpretation as in Bayesian DropOut for CNNs (Gal & Ghahramani, 2015). Specifically, Monte Carlo estimation of GNN outputs can be used to evaluate the predictive posterior uncertainty. An important corollary of this formulation is that any GNN with neighborhood sampling, such as GraphSAGE (Hamilton et al., 2017), could be considered as its corresponding Bayesian approximation.

2. Preliminaries

2.1. Bayesian Neural Networks

Bayesian neural networks (BNNs) aim to capture model uncertainty of DNNs by placing prior distributions over the model parameters to enable posterior updates during DNN training. It has been shown that these Bayesian extensions of traditional DNNs can be robust to over-fitting and provide appropriate prediction uncertainty estimation (Gal & Ghahramani, 2016; Boluki et al., 2020). Often, the standard Gaussian prior distribution is placed over the weights. With random weights $\{\mathbf{W}^{(l)}\}_{l=1}^{L}$, the output prediction given an input $\mathbf{x}$ can be denoted by $\widehat{\mathbf{f}}\big(\mathbf{x}, \{\mathbf{W}^{(l)}\}_{l=1}^{L}\big)$, which is now a random variable in BNNs, enabling uncertainty quantification (UQ).

The key difficulty in using BNNs is that Bayesian inference is computationally intractable. There exist various methods that approximate BNN inference, such as Laplace approximation (MacKay, 1992), sampling-based and stochastic variational inference (Paisley et al., 2012; Rezende et al., 2014; Hajiramezanali et al., 2020; Dadaneh et al., 2020a), Markov chain Monte Carlo (MCMC) (Neal, 2012), and stochastic gradient MCMC (Ma et al., 2015). However, their computational cost is still much higher than the non-Bayesian methods, due to the increased model complexity and slow convergence (Gal & Ghahramani, 2016).

2.2. DropOut as Bayesian Approximation

Dropout is commonly used in training many deep learning models as a way to avoid over-fitting. Using dropout at test time enables UQ with Bayesian interpretation of the network outputs as Monte Carlo samples of its predictive distribution (Gal & Ghahramani, 2016). Various dropout methods have been proposed to multiply the output of each neuron by a random mask drawn from a desired distribution, such as Bernoulli (Hinton et al., 2012; Srivastava et al., 2014) and Gaussian (Kingma et al., 2015; Srivastava et al., 2014). Bernoulli dropout and its extensions are the most commonly used in practice due to their ease of implementation and computational efficiency in existing deep architectures.

2.3. Over-smoothing & Over-fitting in GNNs

It has been shown that graph convolution in graph convolutional neural networks (GCNs) (Kipf & Welling, 2017) is simply a special form of Laplacian smoothing, which mixes the features of a node and its nearby neighbors. Such diffusion operations lead to similar learned representations when the corresponding nodes are close topologically with similar features, thus greatly improving node classification performance. However, it also brings potential concerns of *over-smoothing* (Li et al., 2018). If a GCN is deep with many convolutional layers, the learned representations may be over-smoothed and nodes with different topological and feature characteristics may become indistinguishable. More specifically, by repeatedly applying Laplacian smoothing many times, the node representations within each connected component of the graph will converge to the same values.

Moreover, GCNs, like other deep models, may suffer from *over-fitting* when we utilize an over-parameterized model to fit a distribution with limited training data, where the model we learn fits the training data very well but generalizes poorly to the testing data.

2.4. Stochastic Regularization & Reduction for GNNs

Quickly increasing model complexity and possible over-fitting and over-smoothing when modeling large graphs, as empirically observed in the GNN literature, have been conjectured for the main reason of limited performance

from deep GNNs (Kipf & Welling, 2017; Rong et al., 2019). Several stochastic regularization and reduction methods in GNNs have been proposed to improve the deep GNN performance. For example, *stochastic regularization techniques*, such as DropOut (Srivastava et al., 2014) and DropEdge (Rong et al., 2019), have been used to prevent over-fitting and over-smoothing in GNNs. Sampling-based *stochastic reduction by* random walk neighborhood sampling (Hamilton et al., 2017) and node sampling (Chen et al., 2018) has been deployed in GNNs to reduce the size of input data and thereafter model complexity. Next, we review each of these methods and show that they can be formulated in our proposed adaptive connection sampling framework.

Denote the output of the lth hidden layer in GNNs by $\mathbf{H}^{(l)} = [\mathbf{h}_0^{(l)}, \ldots, \mathbf{h}_n^{(l)}]^T \in \mathbb{R}^{n \times f_l}$ with n being the number of nodes and f_l being the number of output features at the lth layer. Assume $\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{n \times f_0}$ is the input matrix of node attributes, where f_0 is the number of nodes features. Also, assume that $\mathbf{W}^{(l)} \in \mathbb{R}^{f_l \times f_{l+1}}$ and $\sigma(\cdot)$ are the GNN parameters at the lth layer and the corresponding activation function, respectively. Moreover, $\mathcal{N}(v)$ denotes the neighborhood of node v; $\hat{\mathcal{N}}(v) = \mathcal{N}(v) \cup \{v\}$; and $\mathfrak{N}(.)$ is the normalizing operator, i.e., $\mathfrak{N}(\mathbf{A}) = \mathbf{I}_N + \mathbf{D}^{-1/2}\,\mathbf{A}\,\mathbf{D}^{-1/2}$. Finally, $\odot$ represents the Hadamard product.

2.4.1. DropOut (Srivastava et al., 2014)

In a GNN layer, DropOut randomly removes output elements of its previous hidden layer $\mathbf{H}^{(l)}$ based on independent Bernoulli random draws with a constant success rate at each training iteration. This can be formulated as follows:

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathfrak{N}(\mathbf{A})(\mathbf{Z}^{(l)} \odot \mathbf{H}^{(l)})\,\mathbf{W}^{(l)}\right), \qquad (1)$$

where $\mathbf{Z}^{(l)}$ is a random binary matrix, with the same dimensions as $\mathbf{H}^{(l)}$, whose elements are samples of $\text{Bernoulli}(\pi)$. Despite its success in fully connected and convolutional neural networks, DropOut has shown to be ineffectual in GNNs for preventing over-fitting and over-smoothing.

2.4.2. DropEdge (Rong et al., 2019)

DropEdge randomly removes edges from the graph by drawing independent Bernoulli random variables (with a constant rate) at each iteration. More specifically, a GNN layer with DropEdge can be written as follows:

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathfrak{N}(\mathbf{A} \odot \mathbf{Z}^{(l)})\,\mathbf{H}^{(l)}\,\mathbf{W}^{(l)}\right), \qquad (2)$$

Note that here, the random binary mask, i.e. $\mathbf{Z}^{(l)}$, has the same dimensions as $\mathbf{A}$. Its elements are random samples of $\text{Bernoulli}(\pi)$ where their corresponding elements in $\mathbf{A}$ are non-zero and zero everywhere else. It has been shown that the combination of DropOut and DropEdge reaches the best performance in terms of mitigating overfitting in GNNs.

2.4.3. Node Sampling (Chen et al., 2018)

To reduce expensive computation in batch training of GNNs, due to the recursive expansion of neighborhoods across layers, Chen et al. (2018) propose to relax the requirement of simultaneous availability of test data. Considering graph convolutions as integral transforms of embedding functions under probability measures allows for the use of Monte Carlo approaches to consistently estimate the integrals. This leads to an optimal node sampling strategy, FastGCN, which can be formulated as

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathfrak{N}(\mathbf{A})\,\text{diag}(\mathbf{z}^{(l)})\mathbf{H}^{(l)}\,\mathbf{W}^{(l)}\right), \qquad (3)$$

where $\mathbf{z}^{(l)}$ is a random vector whose elements are drawn from $\text{Bernoulli}(\pi)$. This, indeed, is a special case of DropOut, as all of the output features for a node are either completely kept or dropped while DropOut randomly removes some of these related output elements associated with the node.

3. Graph DropConnect

We propose a general stochastic regularization technique for GNNs—Graph DropConnect (GDC)—by adaptive connection sampling, which can be interpreted as an approximation of Bayesian GNNs.

In GDC, we allow GNNs to draw different random masks for each channel and edge independently. More specifically, the operation of a GNN layer with GDC is defined as follows:

$$\mathbf{H}^{(l+1)}[:, j] = \sigma\left(\sum_{i=1}^{f_l} \mathfrak{N}(\mathbf{A} \odot \mathbf{Z}_{i,j}^{(l)})\,\mathbf{H}^{(l)}[:, i]\,\mathbf{W}^{(l)}[i, j]\right), \quad \text{for} \quad j = 1, \ldots, f_{l+1} \qquad (4)$$

where f_l and f_{l+1} are the number of features at layers l and $l+1$, respectively, and $\mathbf{Z}_{i,j}^{(l)}$ is a sparse random matrix (with the same sparsity as $\mathbf{A}$) whose non-zero elements are randomly drawn by $\text{Bernoulli}(\pi_l)$. Note that π_l can be different for each layer for GDC instead of assuming the same constant drop rate for all layers in previous methods.

As shown in (1), (2), and (3), DropOut (Srivastava et al., 2014), DropEdge (Rong et al., 2019), and Node Sampling (Chen et al., 2018) have different sampling assumptions on channels, edges, or nodes, yet there is no clear evidence to favor one over the other in terms of consequent graph analytic performance. In the proposed GDC approach, there is a free parameter $\{\mathbf{Z}_{i,j}^{(l)} \in \{0,1\}^{n \times n}\}_{i=1}^{f_l}$ to adjust the binary mask for the edges, nodes and channels. Thus the proposed GDC model has one extra degree of freedom to incorporate flexible connection sampling.

The previous stochastic regularization techniques can be considered as special cases of GDC. To illustrate that, we

assume $\mathbf{Z}_{i,j}^{(l)}$ are the same for all $j \in \{1, 2, \ldots, f_{l+1}\}$, thus we can omit the indices of the output elements at layer $l+1$ and rewrite (4) as

$$\mathbf{H}^{(l+1)} = \sigma\left(\sum_{i=1}^{f_l} \mathfrak{N}(\mathbf{A} \odot \mathbf{Z}_i^{(l)})\, \mathbf{H}^{(l)}[:, i]\, \mathbf{W}^{(l)}[i, :]\right) \quad (5)$$

Define $\mathbf{J}_n$ as a $n \times n$ all-one matrix. Let $\mathbf{Z}_{DO}^{(l)} \in \{0,1\}^{n \times f_l}$, $\mathbf{Z}_{DE}^{(l)} \in \{0,1\}^{n \times n}$, and $\mathrm{diag}(\mathbf{z}_{NS}^{(l)}) \in \{0,1\}^{n \times n}$ be the random binary matrices corresponding to the ones adopted in DropOut (Srivastava et al., 2014), DropEdge (Rong et al., 2019), and Node Sampling (Chen et al., 2018), respectively. The random mask $\{\mathbf{Z}_i^{(l)} \in \{0,1\}^{n \times n}\}_{i=1}^{f_l}$ in GDC become the same as those of the DropOut when $\mathbf{Z}_i^{(l)} = \mathbf{J}_n\, \mathrm{diag}(\mathbf{Z}_{DO}^{(l)}[:, i])$, the same as those of DropEdge when $\{\mathbf{Z}_i^{(l)}\}_{i=1}^{f_l} = \mathbf{Z}_{DE}^{(l)}$, and the same as those of node sampling when $\{\mathbf{Z}_i^{(l)}\}_{i=1}^{f_l} = \mathbf{J}_n \mathrm{diag}(\mathbf{z}_{NS}^{(l)})$.

3.1. GDC as Bayesian Approximation

In GDC, random masking is applied to the adjacency matrix of the graph to regularize the aggregation steps at each layer of GNNs. In existing Bayesian neural networks, the model parameters, i.e. $\mathbf{W}^{(l)}$, are considered random to enable Bayesian inference based on predictive posterior given training data (Gal et al., 2017; Boluki et al., 2020). Here, we show that connection sampling in GDC can be transformed from the output feature space to the parameter space so that it can be considered as appropriate Bayesian extensions of GNNs.

First, we rewrite (5) to have a node-wise view of a GNN layer with GDC. More specifically,

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\frac{1}{c_v}\Big(\sum_{u \in \hat{\mathcal{N}}(v)} \mathbf{z}_{vu}^{(l)} \odot \mathbf{h}_u^{(l)}\Big)\, \mathbf{W}^{(l)}\right), \quad (6)$$

where c_v is a constant derived from the degree of node v, and $\mathbf{z}_{vu}^{(l)} \in \{0,1\}^{1 \times f_l}$ is the mask row vector corresponding to connection between nodes v and u in three dimensional tensor $\mathcal{Z}^{(l)} = [\mathbf{Z}_1^{(l)}, \ldots, \mathbf{Z}_{f_l}^{(l)}]$. For brevity and without loss of generality, we ignore the constant c_v in the rest of this section. We can rewrite and reorganize (6) to transform the randomness from sampling to the parameter space as

$$\begin{aligned}\mathbf{h}_v^{(l+1)} &= \sigma\left(\Big(\sum_{u \in \hat{\mathcal{N}}(v)} \mathbf{h}_u^{(l)}\, \mathrm{diag}(\mathbf{z}_{vu}^{(l)})\Big)\, \mathbf{W}^{(l)}\right) \\ &= \sigma\left(\sum_{u \in \hat{\mathcal{N}}(v)} \mathbf{h}_u^{(l)}\, \big(\mathrm{diag}(\mathbf{z}_{vu}^{(l)})\, \mathbf{W}^{(l)}\big)\right).\end{aligned} \quad (7)$$

Define $\mathbf{W}_{vu}^{(l)} := \mathbf{z}_{vu}^{(l)}\, \mathbf{W}^{(l)}$. We have:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\sum_{u \in \hat{\mathcal{N}}(v)} \mathbf{h}_u^{(l)}\, \mathbf{W}_{vu}^{(l)}\right). \quad (8)$$

$\mathbf{W}_{vu}^{(l)}$, which pairs the corresponding weight parameter with the edge in the given graph. The operation with GDC in (8) can be interpreted as learning different weights for each of the message passing along edges $e = (u, v) \in \mathcal{E}$ where $\mathcal{E}$ is the union of edge set of the input graph and self-loops for all nodes.

Following the variational interpretation in Gal et al. (2017), GDC can be seen as an approximating distribution $q_\theta(\boldsymbol{\omega})$ for the posterior $p(\boldsymbol{\omega} \mid \mathbf{A}, \mathbf{X})$ when considering a set of random weight matrices $\boldsymbol{\omega} = \{\boldsymbol{\omega}_e\}_{e=1}^{|\mathcal{E}|}$ in the Bayesian framework, where $\boldsymbol{\omega}_e = \{\mathbf{W}_e^{(l)}\}_{l=1}^{L}$ is the set of random weights for the eth edge, $|\mathcal{E}|$ is the number of edges in the input graph, and θ is the set of variational parameters. The Kullback–Leibler (KL) divergence $\mathrm{KL}(q_\theta(\boldsymbol{\omega}) || p(\boldsymbol{\omega}))$ is considered in training as a regularisation term, which ensures that the approximating $q_\theta(\boldsymbol{\omega})$ does not deviate too far from the prior distribution. To be able to evaluate the KL term analytically, the discrete quantised Gaussian can be adopted as the prior distribution as in Gal et al. (2017). Further with the factorization $q_\theta(\boldsymbol{\omega})$ over L layers and $|\mathcal{E}|$ edges such that $q_\theta(\boldsymbol{\omega}) = \prod_l \prod_e q_{\theta_l}(\mathbf{W}_e^{(l)})$ and letting $q_{\theta_l}(\mathbf{W}_e^{(l)}) = \pi_l \delta(\mathbf{W}_e^{(l)} - 0) + (1 - \pi_l)\delta(\mathbf{W}_e^{(l)} - \mathbf{M}^{(l)})$, where $\theta_l = \{\mathbf{M}^{(l)}, \pi_l\}$, the KL term can be written as $\sum_{l=1}^{L} \sum_{e=1}^{|\mathcal{E}|} \mathrm{KL}(q_{\theta_l}(\mathbf{W}_e^{(l)}) \,||\, p(\mathbf{W}_e^{(l)}))$ and approximately

$$\mathrm{KL}(q_{\theta_l}(\mathbf{W}_e^{(l)}) \,||\, p(\mathbf{W}_e^{(l)})) \propto \frac{(1 - \pi_l)}{2} \,||\, \mathbf{M}^{(l)}||^2 - \mathcal{H}(\pi_l),$$

where $\mathcal{H}(\pi_l)$ is the entropy of a Bernoulli random variable with the success rate π_l.

Since the entropy term does not depend on network weight parameters $\mathbf{M}^{(l)}$, it can be omitted when π_l is not optimized. But we learn π_l in GDC, thus the entropy term is important. Minimizing the KL divergence with respect to the drop rate π_l is equivalent to maximizing the entropy of a Bernoulli random variable with probability $1 - \pi_l$. This pushes the drop rate towards 0.5, which may not be desired in some cases where higher/lower drop rate probabilities are more appreciated.

3.2. Variational Inference for GDC

Let's denote $\mathbf{Z}^{(l)} = \{\mathbf{z}_e^{(l)}\}_{e=1}^{|\mathcal{E}|}$ and $\boldsymbol{\omega}^{(l)} = \{\mathbf{W}_e^{(l)}\}_{e=1}^{|\mathcal{E}|}$. For inference of this approximating model with GDC, we assume a factorized variational distribution $q(\boldsymbol{\omega}^{(l)}, \mathbf{Z}^{(l)}) = q(\boldsymbol{\omega}^{(l)} \mid \mathbf{Z}^{(l)})\, q(\mathbf{Z}^{(l)})$. Let the prior distribution $p(\mathbf{W}_e^{(l)} \mid \mathbf{z}_e^{(l)} = 1)$ be a discrete quantised Gaussian, $p(\mathbf{W}_e^{(l)} \mid \mathbf{z}_e^{(l)} = 0)$ be $\delta(\mathbf{W}_e^{(l)} - 0)$, and $p(\boldsymbol{\omega}^{(l)} \mid \mathbf{Z}^{(l)}) = \prod_{e=1}^{\mathcal{E}} p(\mathbf{W}_e^{(l)} \mid \mathbf{z}_e^{(l)})$. Therefore, the KL term can be written

as $\sum_{l=1}^{L} \mathrm{KL}(q(\boldsymbol{\omega}^{(l)}, \mathbf{Z}^{(l)}) \,||\, p(\boldsymbol{\omega}^{(l)}, \mathbf{Z}^{(l)}))$, with

$$\mathrm{KL}\left(q(\boldsymbol{\omega}^{(l)}, \mathbf{Z}^{(l)}) \,||\, p(\boldsymbol{\omega}^{(l)}, \mathbf{Z}^{(l)})\right) \propto \frac{|\mathcal{E}|(1-\pi_l)}{2}||\mathbf{M}^{(l)}||^2 + \sum_{e=1}^{|\mathcal{E}|} \mathrm{KL}\left(q(\mathbf{z}_e^{(l)}) \,||\, p(\mathbf{z}_e^{(l)})\right).$$

The KL term consists of the common weight decay in the non-Bayesian GNNs with the additional KL term $\sum_{e=1}^{|\mathcal{E}|} \mathrm{KL}(q(\mathbf{z}_e^{(l)}) \,||\, p(\mathbf{z}_e^{(l)}))$ acting as a regularization term for $\mathbf{z}_e^{(l)}$. In this GDC framework, the variational inference loss, for node classification for example, can be written as

$$\begin{aligned}\mathcal{L}(\{\mathbf{M}^{(l)}, \pi_l\}_{l=1}^{L}) = \\ \mathbb{E}_{q(\{\boldsymbol{\omega}^{(l)}, \mathbf{Z}^{(l)}\}_{l=1}^{L})}[\log P(Y_o|X, \{\boldsymbol{\omega}^{(l)}, \mathbf{Z}^{(l)}\}_{l=1}^{L})] \\ - \sum_{l=1}^{L} \mathrm{KL}(q(\boldsymbol{\omega}^{(l)}, \mathbf{Z}^{(l)}) \,||\, p(\boldsymbol{\omega}^{(l)}, \mathbf{Z}^{(l)})),\end{aligned} \tag{9}$$

where Y_o denotes the collection of the available labels for the observed nodes. The optimization of (9) with respect to the weight matrices can be done by a Monte Carlo sample, i.e. sampling a random GDC mask and calculating the gradients with respect to $\{\mathbf{M}^{(l)}\}_{l=1}^{L}$ with stochastic gradient descent. It is easy to see that if $\{\pi_l\}_{l=1}^{L}$ are fixed, implementing our GDC is as simple as using common regularization terms on the neural network weights.

We aim to optimize the drop rates $\{\pi_l\}_{l=1}^{L}$ jointly with the weight matrices. This clearly provides more flexibility as all the parameters of the approximating posterior will be learned from the data instead of being fixed *a priori* or treated as hyper-parameters, often difficult to tune. However, the optimization of (9) with respect to the drop rates is challenging. Although the KL term is not a function of the random masks, the commonly adopted reparameterization techniques (Rezende et al., 2014; Kingma & Welling, 2013) are not directly applicable here for computing the expectation in the first term since the drop masks are binary. Moreover, score-function gradient estimators, such as REINFORCE (Williams, 1992; Fu, 2006), possess high variance. One potential solution is continuous relaxation of the drop masks. This approach has lower variance at the expense of introducing bias. Another solution is the direct optimization with respect to the discrete variables by the recently developed Augment-REINFORCE-Merge (ARM) method (Yin & Zhou, 2019; Yin et al., 2020), which has been used in BNNs (Boluki et al., 2020) and information retrieval (Dadaneh et al., 2020b;a). In the next section, we will discuss in detail about our GDC formulation with more flexible beta-Bernoulli prior construction for adaptive connection sampling and how we solve the joint optimization problem for training GNNs with adaptive connection sampling.

4. Variational Beta-Bernoulli GDC

The sampling or drop rate in GDC can be set as a constant hyperparameter as commonly done in other stochastic regularization techniques. In this work, we further enrich GDC with an adaptive sampling mechanism, where the drop rate is directly learned together with GNN parameters given graph data. In fact, in the Bayesian framework, such a hierarchical construct may increase the model expressiveness to further improve prediction and uncertainty estimation performance, as we will show empirically in Section 7.

Note that in this section, for brevity and simplicity we do the derivations for one feature dimension only, i.e. $f_l = 1$. Extending to multi-dimensional features is straightforward as we assume the drop masks are independent across features. Therefore, we drop the feature index in our notations. Inspired by the beta-Bernoulli process (Thibaux & Jordan, 2007), whose marginal representation is also known as the Indian Buffet Process (IBP) (Ghahramani & Griffiths, 2006), we impose a beta-Bernoulli prior to the binary random masks as

$$\begin{aligned} a_e^{(l)} &= z_e^{(l)} a_e, \quad z_e^{(l)} \sim \mathrm{Bernoulli}(\pi_l), \\ \pi_l &\sim \mathrm{Beta}(c/L,\, c(L-1)/L), \end{aligned} \tag{10}$$

where a_e denotes an element of the adjacency matrix $\mathbf{A}$ corresponding to an edge e, and $\hat{a}_e^{(l)}$ an element of the matrix $\hat{\mathbf{A}}^{(l)} = \mathbf{A} \odot \mathbf{Z}^{(l)}$. Such a formulation is known to be capable of enforcing sparsity in random masks (Zhou et al., 2009; Hajiramezanali et al., 2018), which has been shown to be necessary for regularizing deep GNNs as discussed in DropEdge (Rong et al., 2019).

With this hierarchical beta-Bernoulli GDC formulation, inference based on Gibbs sampling can be computationally demanding for large datasets, including graph data (Hasanzadeh et al., 2019). In the following, we derive efficient variational inference algorithm(s) for learnable GDC.

To perform variational inference for GDC random masks and the corresponding drop rate at each GNN layer together with weight parameters, we define the variational distribution as $q(\mathbf{Z}^{(l)}, \pi_l) = q(\mathbf{Z}^{(l)} \,|\, \pi_l) q(\pi_l)$. We define $q(\pi_l)$ to be Kumaraswamy distribution (Kumaraswamy, 1980); as an alternative to the beta prior factorized over lth layer

$$q(\pi_l; a_l, b_l) = a_l b_l \pi_l^{a_l - 1}(1 - \pi_l^{a_l})^{b_l - 1}, \tag{11}$$

where a_l and b_l are greater than zero. Knowing π_l the edges are independent, thus we can rewrite $q(\mathbf{Z}^{(l)} \,|\, \pi_l) = \prod_{e=1}^{|\mathcal{E}|} q(z_e^{(l)} \,|\, \pi_l)$. We further put a Bernoulli distribution with parameter π_l over $q(\mathbf{z}_e^{(l)}|\pi_l)$. The KL divergence term

can be written as

$$\mathrm{KL}\left(q(\mathbf{Z}^{(l)},\pi_l)\,||\,p(\mathbf{Z}^{(l)},\pi_l)\right) = \sum_{e=1}^{|\mathcal{E}|}\mathrm{KL}\left(q(z_e^{(l)}\,|\,\pi_l)\,||\,p(z_e^{(l)}\,|\,\pi_l)\right)+\mathrm{KL}\left(q(\pi_l)\,||\,p(\pi_l)\right).$$

While the first term is zero due to the identical distributions, the second term can be computed in closed-form as

$$\mathrm{KL}\left(q(\pi_l)\,||\,p(\pi_l)\right) = \frac{a_l-c/L}{a_l}\left(-\gamma-\Psi(b_l)-\frac{1}{b_l}\right)+\log\frac{a_l b_l}{c/L}-\frac{b_l-1}{b_l},$$

where γ is the Euler-Mascheroni constant and $\Psi(\cdot)$ is the digamma function.

The gradient of the KL term in (9) can easily be calculated with respect to the drop parameters. However, as mentioned in the previous section, due to the discrete nature of the random masks, we cannot directly apply reparameterization technique to calculate the gradient of the first term in (9) with respect to the drop rates (parameters). One way to address this issue is to replace the discrete variables with a continuous approximation. We impose a concrete distribution relaxation (Jang et al., 2016; Gal et al., 2017) for the Bernoulli random variable $z_{uv}^{(l)}$, leading to an efficient optimization by sampling from simple sigmoid distribution which has a convenient parametrization

$$\tilde{z}_e^{(l)} = \mathrm{sigmoid}\left(\frac{1}{t}\big(\log\big(\frac{\pi_l}{1-\pi_l}\big)+\log\big(\frac{u}{1-u}\big)\big)\right), \tag{12}$$

where $u\sim\mathrm{Unif}[0,1]$ and t is temperature parameter of relaxation. We can then use stochastic gradient variational Bayes to optimize the variational parameters a_l and b_l.

Although this approach is simple, the relaxation introduces bias. Our other approach is to directly optimize the variational parameters using the original Bernoulli distribution in the formulation as in Boluki et al. (2020). We can calculate the gradient of the variational loss with respect to $\boldsymbol{\alpha}=\{\mathrm{logit}(1-\pi_l)\}_{l=1}^L$ using ARM estimator , which is unbiased and has low variance, by performing two forward passes as

$$\nabla_{\boldsymbol{u}}\mathcal{L}(\alpha) = \mathbb{E}_{u\sim\prod_{l=1}^L\prod_{e=1}^{|\mathcal{E}|}\mathrm{Unif}[0,1](u_e^{(l)})}\Big[\big(\mathcal{L}(\{\mathbf{M}^{(l)}\}_{l=1}^L, 1_{[\boldsymbol{u}>\sigma(-\boldsymbol{\alpha})]})-\mathcal{L}(\{\mathbf{M}^{(l)}\}_{l=1}^L,1_{[\boldsymbol{u}<\sigma(\boldsymbol{\alpha})]})\big)\big(\boldsymbol{u}-\frac{1}{2}\big)\Big],$$

where $\mathcal{L}(\{\mathbf{M}^{(l)}\}_{l=1}^L,1_{[\boldsymbol{u}<\sigma(\boldsymbol{\alpha})]})$ denotes the loss obtained by setting $\mathbf{Z}^{(l)} = 1_{[\boldsymbol{u}^{(l)}<\sigma(\boldsymbol{\alpha}_l)]} := \big(1_{[u_1^{(l)}<\sigma(\alpha_l)]},\ldots,1_{[u_{|\mathcal{E}|}^{(l)}<\sigma(\alpha_l)]}\big)$ for $l=1,\ldots,L$. The gradient with respect to $\{a_l,b_l\}_{l=1}^L$ can then be calculated by using the chain rule and the reparameterization for $\pi_l=(1-u^{\frac{1}{b_l}})^{\frac{1}{a_l}}, u\sim\mathrm{Unif}[0,1]$.

It is worth noting that although the beta-Bernoulli DropConnect with ARM is expected to provide better performance due to the more accurate gradient estimates, it has slightly higher computational complexity as it requires two forward passes.

5. Connection to Random Walk Sampling

Various types of random walk have been used in graph representation learning literature to reduce the size of input graphs. In GNNs, specifically in GraphSAGE (Hamilton et al., 2017), random walk sampling has been deployed to reduce the model complexity for very large graphs. One can formulate a GNN layer with random walk sampling as follows:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\Big(\sum_{u\in\hat{\mathcal{N}}(v)}(z_{vu}^{(l)}\,|\,\mathbf{Z}^{(l-1)})\,\mathbf{h}_u^{(l)}\Big)\,\mathbf{W}^{(l)}\right). \tag{13}$$

Here, $\mathbf{Z}^{(l)}$ is the same as the one in DropEdge except that it is dependent on the masks from the previous layer. This is due to the fact that random walk samples for each node are connected subgraphs.

In this setup, we can decompose the variational distribution of the GDC formulation in an autoregressive way. Specifically, here we have $q(z_{uv}^{(l)}|\mathbf{Z}^{(l-1)}) = \mathrm{Bernoulli}(\pi_l)1_{\sum_{u\in\hat{\mathcal{N}}(v)}z_{vu}^{(l-1)}>0}$. With fixed Bernoulli parameters, we can calculate the gradients for the weight matrices with Monte Carlo integration. Learning Bernoulli parameters is challenging and does not allow direct application of ARM due to the autoregressive structure of the variational posterior. We leave sequential ARM for future study.

Corollary 1 *Any graph neural network with random walk sampling, such as GraphSAGE, is an approximation of a Bayesian graph neural network as long as outputs are calculated using Monte Carlo sampling.*

6. Sampling Complexity

The number of random samples needed for variational inference in GDC, (4), at each layer of a GNN is $|\mathcal{E}|\times f_l\times f_{l+1}$. This number would reduce to $|\mathcal{E}|\times f_l$ in the constrained version of GDC as shown in (5). These numbers, potentially, could be very high specially if the size of the graph or the number of filters are large, which could increase the space complexity and computation time. To circumvent this issue, we propose to draw a single sample for a block of features as oppose to drawing a new sample for every single feature. This would reduce the number of required samples to $|\mathcal{E}|\times\mathrm{nb}$ with nb being the number of blocks. In our experiments, we have one block in the first layer and two blocks in layers after that. In our experiments, we keep

Table 1. Semi-supervised node classification accuracy of GCNs with our adaptive connection sampling and baseline methods.

Method	Cora		Citeseer		Cora-ML	
	2 layers	4 layers	2 layers	4 layers	2 layers	4 layers
GCN-DO	80.98 ± 0.48	78.24 ± 2.4	70.44 ± 0.39	64.38 ± 0.90	83.45 ± 0.73	81.51 ± 1.01
GCN-DE	78.36 ± 0.92	73.40 ± 2.07	70.52 ± 0.75	57.14 ± 0.90	83.30 ± 1.37	68.89 ± 3.37
GCN-DO-DE	80.58 ± 1.19	79.20 ± 1.07	70.74 ± 1.23	64.84 ± 0.98	83.61 ± 0.83	81.21 ± 1.53
GCN-BBDE	$\mathbf{81.58} \pm 0.49$	$\mathbf{80.42} \pm 0.25$	$\mathbf{71.46} \pm 0.55$	$\mathbf{68.58} \pm 0.88$	$\mathbf{84.62} \pm 1.70$	$\mathbf{84.73} \pm 0.52$
GCN-BBGDC	$\mathbf{81.80} \pm 0.99$	$\mathbf{82.20} \pm 0.92$	$\mathbf{71.72} \pm 0.48$	$\mathbf{70.00} \pm 0.36$	$\mathbf{85.43} \pm 0.70$	$\mathbf{85.52} \pm 0.83$

the order of features the same as the original input files, and divide them into nb groups with the equal number of features.

While in our GDC formulation, as shown in (4) and (5), the normalization $\mathfrak{N}(\cdot)$ is applied after masking, one can multiply the randomly drawn mask with the pre-computed normalized adjacency matrix. This relaxation reduces the computation time and has negligible effect on the performance based on our experiments. An extension to the GDC sampling strategy is asymmetric sampling where the mask matrix $\mathbf{Z}$ could be asymmetric. This would increase the number of samples by a factor of two; however it increases the model flexibility. In our experiments, we have used asymmetric masks and multiplied the mask with the normalized adjacency matrix.

7. Numerical Results

We test the performance of our adaptive connection sampling framework, learnable GDC, on semi-supervised node classification using real-world citation graphs[1]. In addition, we compare the uncertainty estimates of predictions by Monte Carlo beta-Bernoulli GDC and Monte Carlo Dropout. We also show the performance of GDC compared to existing methods in alleviating the issue of over-smoothing in GNNs. Furthermore, we investigate the effect of the number of blocks on the performance of GDC. We have also investigated learning separate drop rates for every edge in the network, i.e. *local* GDC, which is included in the supplementary materials.

7.1. Semi-supervised Node Classification

7.1.1. Datasets and Implementation Details

We conducted extensive experiments for semi-supervised node classification with real-world citation datasets. We consider *Cora*, *Citeseer* and *Cora-ML* datasets, and preprocess and split them same as Kipf & Welling (2017) and Bojchevski & Gunnemann (2018). We train beta-Bernoulli GDC (BBGDC) models for 2000 epochs with a learning rate of 0.005 and a validation set used for early stopping. All of the hidden layers in our implemented GCNs have 128 dimensional output features. We use 5×10^{-3}, 10^{-2}, and 10^{-3} as L2 regularization factor for Cora, Citeseer, and Cora-ML, respectively. For the GCNs with more than 2 layers, we use warm-up during the first 50 training epochs to gradually impose the beta-Bernoulli KL term in the objective function. The temperature in the concrete distribution is set to 0.67. For a fair comparison, the number of hidden units are the same in the baselines and their hyper-parameters are hand-tuned to achieve their best performance. Performance is reported by the average accuracy with standard deviation based on 5 runs on the test set. The dataset statistics as well as more implementation details are included in the supplementary materials.

7.1.2. Discussion

Table 1 shows that BBGDC outperforms the state-of-the-art stochastic regularization techniques in terms of accuracy in all benchmark datasets. DO and DE in the table stand for DropOut and DropEdge, respectively. Comparing GCN-DO and GCN-DE, we can see that DropEdge alone is less effective than DropOut in overcoming over-smoothing and over-fitting in GCNs. The difference between accuracy of GCN-DO and GCN-DE is more substantial in deeper networks (5% in Cora, 7% in Citeseer, and 13% in Cora-ML), which further proves the limitations of DE. Among the baselines, combination of DO and DE shows the best performance allowing to have deeper models. However, this is not always true. For example in Citeseer, 4-layer GCN shows significant decrease in performance compared to 2-layer GCN.

To show the advantages of learning the drop rates as well as the effect of hierarchical beta-Bernoulli construction, we have also evaluated beta-Bernoulli DropEdge (BBDE) with the concrete approximation, in which the edge drop rate at *each layer* is learned using the same beta-Bernoulli hierarchical construction as GDC. We see that GCN with BBDE, without any DropOut, performs better than both GCNs with DE and DO-DE. By comparing GCN with BBDE and GCN

[1]Our code is available at https://github.com/armanihm/GDC

Table 2. Accuracy of ARM optimization-based variants of our proposed method in semi-supervised node classification.

Method	**Cora** (4 layers)	**Citeseer** (4 layers)
GCN-BDE-ARM	79.95 ± 0.79	67.90 ± 0.15
GCN-BBDE-ARM	81.74 ± 0.75	69.82 ± 0.58
GCN-BBGDC-ARM	82.46 ± 0.26	70.52 ± 0.44

with BBGDC, it is clear that the improvement is not only due to learnable sampling rate but also the increased flexibility of GDC compared to DropEdge. We note that GCN-BBGDC is the only method for which the accuracy improves by increasing the number of layers except in Citeseer.

7.1.3. Concrete relaxation versus ARM

To investigate the effect of direct optimization of the variational loss with respect to the drop parameters with ARM vs relaxation of the discrete random variables with concrete, we construct three ARM optimization-based variants of our method: Learnable Bernoulli DropEdge with ARM gradient estimator (BDE-ARM) where the edge drop rate of the Bernoulli mask at each layer is directly optimized; beta-Bernoulli DropEdge with ARM (BBDE-ARM); and beta-Bernoulli GDC with ARM (BBGDC-ARM). We evaluate these methods on the 4-layer GCN setups where significant performance improvement compared with the baselines has been achieved by BBDE and GDC with concrete relaxation. Comparing the performance of BBDE-ARM and BBGDC-ARM in Table 2 with the corresponding models with concrete relaxation, suggests further improvement when the drop parameters are directly optimized. Moreover, BDE-ARM, which optimizes the parameters of the Bernoulli drop rates, performs better than DO, DE, and DO-DE.

7.2. Uncertainty Quantification

To evaluate the quality of uncertainty estimates obtained by our model, we use the Patch Accuracy vs Patch Uncertainty (PAvPU) metric introduced in (Mukhoti & Gal, 2018). PAvPU combines $p(\text{accurate}|\text{certain})$, i.e. the probability that the model is accurate on its output given that it is confident on the same, $p(\text{certain}|\text{inaccurate})$, i.e. the probability that the model is uncertain about its output given that it has made a mistake in its prediction, into a single metric. More specifically, it is defined as $\text{PAvPU} = (n_{ac} + n_{iu})/(n_{ac} + n_{au} + n_{ic} + n_{iu})$, where n_{ac} is the number of accurate and certain predictions, n_{au} is the number of accurate and uncertain predictions, n_{ic} is the number of inaccurate and certain predictions, and n_{iu} is the number of inaccurate and uncertain predictions. Higher PAvPU means that certain predictions are accurate and inaccurate predictions are uncertain.

We here demonstrate the results for uncertainty estimates for a 4-layer GCN-DO and a 4-layer GCN-BBGDC with random initialization for semi-supervised node classification on Cora. We have evaluated PAvPU using 20 Monte Carlo samples for the test set where we use predictive entropy as the uncertainty metric. The results are shown in Figure 1. It can be seen that our proposed model consistently outperforms GCN-DO on every uncertainty threshold ranging from 0.5 to 1 of the maximum predictive uncertainty. While Figure 1 depicts the results based on one random initialization, other initializations show the same trend.

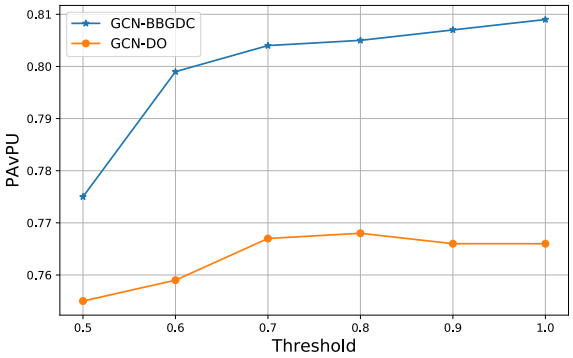

Figure 1. Comparison of uncertainty estimates in PAvPU by a 4-layer GCN-BBGDC with 128-dimensional hidden layers and a 4-layer GCN-DO 128-dimensional hidden layers on Cora.

7.3. Over-smoothing and Over-fitting

To check how GDC helps alleviate over-smoothing in GCNs, we have tracked the total variation (TV) of the outputs of hidden layers during training. TV is a metric used in the graph signal processing literature to measure the smoothness of a signal defined over nodes of a graph (Chen et al., 2015). More specifically, given a graph with the adjacency matrix $\mathbf{A}$ and a signal $\mathbf{x}$ defined over its nodes, TV is defined as $\text{TV}(\mathbf{x}) = \|\mathbf{x} - (1/|\lambda_{max}|)\mathbf{A}\,\mathbf{x}\|_2^2$, where λ_{max} denotes the eigenvalue of A with largest magnitude. Lower TV shows that the signal on adjacent nodes are closer to each other, indicating possible over-smoothing.

We have compared the TV trajectories of the hidden layer outputs in a 4-layer GCN-BBGDC and a 4-layer GCN-DO normalized by their Frobenius norm, depicted in Figure 2(a). It can be seen that, in GCN-DO, while the TV of the first layer is slightly increasing at each training epoch, the TV of the second hidden layer decreases during training. This, indeed, contributed to the poor performance of GCN-DO. On the contrary, the TVs of both first and second layers in GCN-BBGDC is increasing during training. Not only this robustness is due to the dropping connections in GDC framework, but also is related to its learnable drop rates.

With such promising results showing less over-smoothing

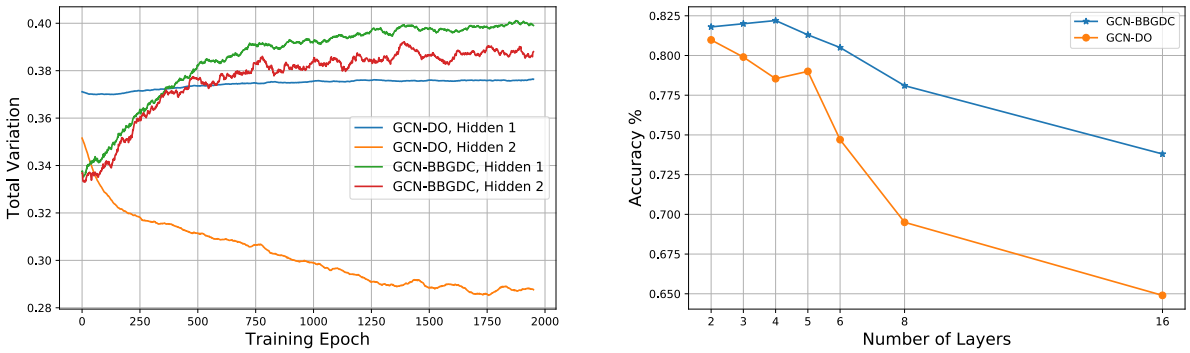

Figure 2. From left to right: a) Total variation of the hidden layer outputs during training in a 4-layer GCN-BBGDC with 128-dimensional hidden layers and a 4-layer GCN-DO 128-dimensional hidden layers on Cora; b) Comparison of node classification accuracy for GCNs with a different number of hidden layers using different stochastic regularization methods. All of the hidden layers are 128 dimensional.

with BBGDC, we further investigate how our proposed method works in deeper networks. We have checked the accuracy of GCN-BBGDC with a various number of 128-dimensional hidden layers ranging from 2 to 16. The results are shown in Figure 2(b). The performance improves up to the GCN with 4 hidden layers and decreases after that. It is important to note that even though the performance drops by adding the 5-th layer, the degree to which it decreases is far less than competing methods. For example, the node classification accuracy with GCN-DO quickly drops to 69.50% and 64.5% with 8 and 16 layers. In addition, we should mention that the performance of GCN-DO only improves from two to three layers. This, indeed, proves GDC is a better stochastic regularization framework for GNNs in alleviating over-fitting and over-smoothing, enabling possible directions to develop deeper GNNs.

7.4. Effect of Number of Blocks

In GDC for every pair of input and output features, a separate mask for the adjacency matrix should be drawn. However, as we discussed in Section 6, this demands large memory space. We circumvented this problem by drawing a single mask for a block of features. While we used only two blocks in our experiments presented so far, we here investigate the effect of the number of blocks on the node classification accuracy. The performance of 128-dimensional 4-layer GCN-BBGDC with 2, 16, and 32 blocks are shown in Table 3. As can be seen, the accuracy improves as the number of blocks increases. This is due to the fact that increasing the number of blocks increases the flexibility of GDC. The choice of the number of blocks is a factor to consider for the trade off between the performance and memory usage as well as computational complexity.

Table 3. Accuracy of 128-dimensional 4-layer GCN-BBGDC with different number of blocks on Cora in semi-supervised node classification.

Method	2 blocks	16 blocks	32 blocks
GCN-BBGDC	82.2	83.0	83.3

8. Conclusion

In this paper, we proposed a unified framework for adaptive connection sampling in GNNs that generalizes existing stochastic regularization techniques for training GNNs. Our proposed method, Graph DropConnect (GDC), not only alleviates over-smoothing and over-fitting tendencies of deep GNNs, but also enables learning with uncertainty in graph analytic tasks with GNNs. Instead of using fixed sampling rates, our GDC technique parameters can be trained jointly with GNN model parameters. We further show that training a GNN with GDC is equivalent to an approximation of training Bayesian GNNs. Our experimental results shows that GDC boosts the performance of GNNs in semi-supervised classification task by alleviating over-smoothing and over-fitting. We further show that the quality of uncertainty derived by GDC is better than DropOut in GNNs.

Acknowledgements

The presented materials are based upon the work supported by the National Science Foundation under Grants ENG-1839816, IIS-1848596, CCF-1553281, IIS-1812641, IIS-1812699, and CCF-1934904.

References

Bojchevski, A. and Gunnemann, S. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

Boluki, S., Ardywibowo, R., Dadaneh, S. Z., Zhou, M., and

Qian, X. Learnable Bernoulli dropout for Bayesian deep learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 3905–3916, 2020.

Chen, J., Ma, T., and Xiao, C. FastGCN: Fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Chen, S., Sandryhaila, A., Moura, J. M., and Kovačević, J. Signal recovery on graphs: Variation minimization. *IEEE Transactions on Signal Processing*, 63(17):4609–4624, 2015.

Dadaneh, S. Z., Boluki, S., Yin, M., Zhou, M., and Qian, X. Pairwise supervised hashing with Bernoulli variational auto-encoder and self-control gradient estimator. In *The Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020a.

Dadaneh, S. Z., Boluki, S., Zhou, M., and Qian, X. Arsm gradient estimator for supervised learning to rank. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3157–3161, 2020b.

Fu, M. C. Gradient estimation. *Handbooks in operations research and management science*, 13:575–616, 2006.

Gal, Y. and Ghahramani, Z. Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*, 2015.

Gal, Y. and Ghahramani, Z. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

Gal, Y., Hron, J., and Kendall, A. Concrete dropout. In *Advances in neural information processing systems*, pp. 3581–3590, 2017.

Ghahramani, Z. and Griffiths, T. L. Infinite latent feature models and the Indian buffet process. In *Advances in neural information processing systems*, pp. 475–482, 2006.

Hajiramezanali, E., Dadaneh, S. Z., Karbalayghareh, A., Zhou, M., and Qian, X. Bayesian multi-domain learning for cancer subtype discovery from next-generation sequencing count data. In *Advances in Neural Information Processing Systems*, pp. 9115–9124, 2018.

Hajiramezanali, E., Hasanzadeh, A., Duffield, N., Narayanan, K. R., Zhou, M., and Qian, X. Variational graph recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2019.

Hajiramezanali, E., Hasanzadeh, A., Duffield, N., Narayanan, K., Zhou, M., and Qian, X. Semi-implicit stochastic recurrent neural networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3342–3346. IEEE, 2020.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Hasanzadeh, A., Hajiramezanali, E., Duffield, N., Narayanan, K. R., Zhou, M., and Qian, X. Semi-implicit graph variational auto-encoders. In *Advances in Neural Information Processing Systems*, 2019.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

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

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pp. 2575–2583, 2015.

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

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Kumaraswamy, P. A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, 46(1-2):79–88, 1980.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Ma, Y.-A., Chen, T., and Fox, E. A complete recipe for stochastic gradient MCMC. In *Advances in Neural Information Processing Systems*, pp. 2917–2925, 2015.

MacKay, D. J. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

Mukhoti, J. and Gal, Y. Evaluating bayesian deep learning methods for semantic segmentation. *arXiv preprint arXiv:1811.12709*, 2018.

Neal, R. M. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Paisley, J., Blei, D., and Jordan, M. Variational Bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*, 2012.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Rong, Y., Huang, W., Xu, T., and Huang, J. DropEdge: Towards the very deep graph convolutional networks for node classification, 2019.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Thibaux, R. and Jordan, M. I. Hierarchical beta processes and the Indian buffet process. In *Artificial Intelligence and Statistics*, pp. 564–571, 2007.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Yin, M. and Zhou, M. ARM: Augment-REINFORCE-merge gradient for stochastic binary networks. In *International Conference on Learning Representations*, 2019.

Yin, M., Ho, N., Yan, B., Qian, X., and Zhou, M. Probabilistic best subset selection by gradient-based optimization. *arXiv preprint arXiv:2006.06448*, 2020.

Zhang, Y., Pal, S., Coates, M., and Ustebay, D. Bayesian graph convolutional neural networks for semi-supervised classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5829–5836, 2019.

Zhou, M., Chen, H., Ren, L., Sapiro, G., Carin, L., and Paisley, J. W. Non-parametric Bayesian dictionary learning for sparse image representations. In *Advances in neural information processing systems*, pp. 2295–2303, 2009.