Bayesian Graph Neural Networks

Alexandru I. Oarga Hategan
Faculty of Mathematics and Computer Science
University of Barcelona
Gran Via de les Corts Catalanes, 585, Barcelona
aoargaha35@alumnes.ub.edu

Carmen Casas Herce
Faculty of Mathematics and Computer Science
University of Barcelona
Gran Via de les Corts Catalanes, 585, Barcelona
ccasashe86@alumnes.ub.edu

Manuel A. Hernández Alonso
Faculty of Mathematics and Computer Science
University of Barcelona
Gran Via de les Corts Catalanes, 585, Barcelona
manuelheralo@gmail.com

Víctor Pérez Fayos
Faculty of Mathematics and Computer Science
University of Barcelona
Gran Via de les Corts Catalanes, 585, Barcelona
vperezfa9@alumnes.ub.edu

Abstract

Graph Neural Networks (GNN) is the fundamental algorithm for learning graph-structured data. Nonetheless, they are limited by several factors regarding noisy data and base assumptions that may not hold. The main objective of this research is to find alternatives based on Bayesian rules and inferences. Previous research has found several architectures and structures based on Bayesian statistics such as: Bayesian Graph Convolutional Neural Networks (BGCN), Graph DropConnect model (GDC), Graph Variational Auto-Encoders (GVAE), and Graph structure Estimation Networks (GEN). We used the Cora dataset to validate and test these models in similar experimental setups, where we used the models on (a) a semi-supervised node classification task, and (b) an edge-linking prediction task. In this research, similar results to those found in their respective papers with only a few points lower in some cases due to not tuning further the hyperparameters to replicate the results. In conclusion, the field of GNNs is still in development and different approaches are being explored, this paper has allowed us to explore new architectures based on Bayesian statistics in order to achieve better performances and results in graph-based tasks. All the code used can be found in `https://github.com/alexOarga/bayesian-graph-neural-networks`:

1 Introduction

Graph Neural Networks (GNN), first proposed in [20], have become the reference algorithm for learning on graph-structured data. Simple GNNs however have some limitations by themselves, some simple graph structures cannot be distinguished, making assumptions on the graph such as homophily [6] and homogeneity that may not always hold. These GNNs are also not robust to noise in the data, such as random edge deletion/addition or node feature perturbation. Additionally, some other problems may present when working on more complex models as they may lack the complexity to capture latent features and structures.

To evolve GNNs and solve these problems, a variety of Bayesian GNNs have been recently proposed in the literature, following different assumptions about Bayesian statistics. In this work, we have selected Bayesian GNNs that: (1) assume that the weights of the model are random variables, such as Bayesian Graph Convolutional Neural Networks and Beta-the Bernoulli Graph DropConnect, a variation of the Graph DropConnect model, (2) assume nodes and graphs are generated by latent variables, such as Graph Variational Auto-Encoders, and (3) try to identify the optimal graph structure from the posterior distributions generated by observations of the graph, such as Graph structure Estimation Networks.

The rest of the paper is organised as follows: Section 2 introduces preliminary definitions, Section 3 presents different models of Bayesian graph neural networks, Section 4 presents the results obtained with the models on the Cora [1] graph dataset, and Section 5 ends with the main remarks about the work carried out.

2 Preliminary definitions

This section introduces preliminary definitions that are used throughout the whole paper. In particular, Graph Neural Networks (GNNs) and Graph Convolutional Networks (GCN). Additionally, Monte Carlo Dropout is defined, stochastic regularisation techniques are recalled and a definition of Variational Auto-Encoders (VAE) is provided.

2.1 Graph Neural Networks

Graph Neural Networks (GNN) [20, 16, 11] are machine learning models that learn on data that is accompanied by a graph structure. GNNs are composed of layers of message-passing networks. In each layer, the embedding vector h_i of node i is computed from the aggregation of the embeddings of their neighbour nodes $\mathcal{N}(i)$ of the previous layer (see Figure 1). The initial embedding vector is usually the input feature vector that each node is given. As described in [27], a general k-th GNN layer can be defined formally as:

$$h_v^{(k+1)} = \text{AGG}\left(\left\{\text{ACT}\left(\mathbf{W}^{(k)} h_u^{(k)} + b^{(k)}\right), u \in \mathcal{N}(v)\right\}\right) \tag{1}$$

where $h_v^{(k)}$ is the k-th layer embedding of node v, $\mathbf{W}^{(k)}$ and $b^{(k)}$ are the trainable weight matrix and bias respectively, ACT is an activation function and AGG is a commutative aggregation function such as maximisation, summation or mean. The different variations proposed to GNNs have shown to be very effective in learning on graphs data [7, 23, 16]. However, GNNs are known to suffer from limited expressivity [25], over-smoothing, this is, converging to a single solution after many layers [15], and over-squashing, this is, losing information from long-range nodes due to bottlenecks in the graph [2].

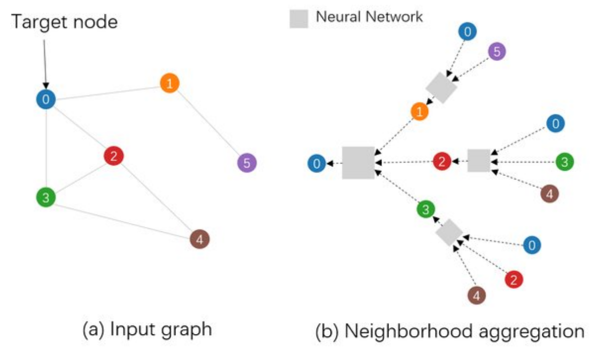

Figure 1: Each GNN layer computes a node representation by aggregating its neighbours' representations. Source: [9].

2.1.1 Graph Convolutional Networks

Currently, numerous models of GNNs have been presented in the literature, as properly modelling data on graphs is still an open field of study [27]. We focus on Graph Convolutional Networks (GCNs) as they are acknowledged to be one of the first baseline models of GNN, where the model leverages the features of the nodes to make predictions in a semi-supervised classification way.

Formally, a *Graph Convolutional Network* [13, 14] is a neural network where the features of a node $h_i^{(l)}$, from layer l are combined with the features of its neighbours generating the next layer embedding $h_i^{(l+1)}$:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \epsilon \mathcal{N}_i} W h_j^{(l)}\right) \tag{2}$$

Where $\mathcal{N}_i$ are all the neighbours of node x (including itself), and σ is the activation function. Generally, $h_i^{(0)}$ is equal to the initial features of each node. As the size of the neighbourhood is not fixed, it is necessary to normalise each embedding using the degree of the node. Usually a *symmetric normalisation factor*, that takes into account the degree of both the main node and each of the neighbours, is used instead of the degree of a single node, in order to balance the influence in general.

$$h_i^{(l+1)} = \sigma\left(\sum_{j \epsilon \mathcal{N}_i} \frac{1}{\sqrt{deg(i)}\sqrt{deg(j)}} W h_j^{(l)}\right) \tag{3}$$

Following a matrix notation, a GCN layer is defined as:

$$H^{(l+1)} = \sigma\left(\tilde{A} H^{(l)} W^{(l)}\right) \tag{4}$$

where $\tilde{A} = A + I$, is the adjacency matrix of the undirected graph with added self-loops. The degree normalised version is given by:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right) \tag{5}$$

with $\tilde{D}_{ii} = \sum_j \tilde{A}_{jj}$. A function of the model [17] at layer l is thus given by:

$$\phi(x_i, X_{\mathcal{N}_i}, l) = H^{(l)} \tag{6}$$

where $x_i \in X$ are the initial features of node i. Adding more layers to the architecture of the GCN implies aggregating the features of higher-order neighbours, although adding too many causes that all embeddings are almost the same, as all take into account a lot of common nodes. Usually, networks of 2 or 3 layers have been shown to obtain the best results [17].

2.2 Monte Carlo Dropout

The GNNs models introduced in the previous section can be extended to implement some assumptions that come from Bayesian Learning. Instead of working with the given graph dataset as the ground truth, we can assume that a sample comes from a distribution of graphs. In this sense, an input graph is a random variable of this unknown distribution [17].

If we consider an observed graph $\mathcal{G}_{obs} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ the set of edges, $\mathbf{X}$ is the feature nodes matrix, $\mathbf{Y}_{obs}$ the set of labels that are known and $\mathbf{W}$ are the weights of the neural network, that form a random variable, we can compute the posterior distribution of the weights of the model as follows [17], :

$$p(\mathbf{W}|\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs}) = \frac{p(\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs}|\mathbf{W})p(\mathbf{W})}{p(\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs})} \tag{7}$$

and the marginal likelihood is then given by:

$$p(\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs}) = \int p(\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs}|\mathbf{W})p(\mathbf{W})d\mathbf{W} \tag{8}$$

It can be seen however that these equations are very expensive to solve, so an approximation needs to be done in order to simplify the calculus. Several methods have been implemented to approximate the posterior distribution function, such as Variational Methods or using Markov Chain Monte Carlo. [28]. However, one of the methods used in the models studied is the *Monte Carlo Dropout*. The idea behind this procedure is to train several times a NN with dropout (each time the dropout will affect to different neurons) and then compute the mean and the variance of the outputs. These layers where the dropout has been applied are present both in training and test processes [5].

Although it is an approximation that has some limitations, especially when the target distribution is too complex, it is widely used because it is simple to understand and implement and the computation time that requires is little [17]. We can also transform the equation (7) as necessary depending on the properties we want to predict from the graph. If we want to predict the category a node belongs to, and compute a matrix of label probabilities $\mathbf{Z}$, the equation would be [17]:

$$p(\mathbf{Z}|\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs}) = \int p(\mathbf{Z}|\mathbf{X}, \mathbf{W}, \mathcal{G}_{obs})p(\mathbf{W}|\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs})p(\mathcal{G}|\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs})d\mathbf{W}d\mathcal{G} \tag{9}$$

The idea to simplify the integral is to directly sample predictions $\mathbf{z}$ from the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs})$ and use this input to train a neural network applying MC dropout.

2.3 Stochastic regularization for GNNS

As it was mentioned earlier in this section, GNNs can suffer from over-smoothing and over-squashing. Additionally, increasing models complexity can also lead to overfitting. To address these many works have proposed to use stochastic regularization techniques. Here we present three of them: (1) DropOut [21], (2) DropEdge [19], and (3) Node Sampling [4]. The model Graph DropConnect, which is later presented in this work is based on these techniques.

2.3.1 DropOut

In GNNs, *DropOut* [21] consists of randomly removing elements from the previous layer features. Formally, DropOut for a GNN layer is defined as:

$$\mathbf{H}^{(k+1)} = \sigma\left(\mathfrak{N}(\mathbf{A})(\mathbf{Z}_i^{(l)} \odot \mathbf{H}^{(l)})\mathbf{W}^{(l)}\right) \tag{10}$$

where $\mathbf{Z}_i^{(l)}$ is a random binary matrix, $\odot$ is the Hadaamard product, $\mathfrak{N}(\mathbf{A})$ is the normalizing operator (i.e. $\mathfrak{N}(\mathbf{A}) = I + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$), $\mathbf{H}^{(l)}$ is the nodes features matrix at layer l, and $\mathbf{W}^{(l)}$ is the weight matrix at layer l.

2.3.2 DropEdge

DropEdge [19] consists of randomly deleting edges from a graph by sampling from a Bernoulli distribution (with constant rate). Formally, DropEdge is defined as:

$$\mathbf{H}^{(k+1)} = \sigma\left((\mathfrak{N}(\mathbf{A}) \odot \mathbf{Z}_i^{(l)})\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{11}$$

where the elements of matrix $\mathbf{Z}_i^{(l)}$ are sampled from a Bernoulli distribution.

2.3.3 Noise Sampling

Noise Sampling [4] is a technique that reduces the number of neighbour node samples. The reasoning behind the technique is out of the scope of this work and we refer the reader to the original work for details. Formally, Noise Sampling is defined as:

$$\mathbf{H}^{(k+1)} = \sigma\left(\mathfrak{N}(\mathbf{A})diag(\mathbf{z}_i^{(l)})\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{12}$$

where $\mathbf{z}_i^{(l)}$ is a random binary vector drawn from a Bernoulli distribution. Note that this is a special case of DropOut, because all the features of a node are either kept or removed.

2.4 Variational Autoencoders

Given a dataset $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$ of N samples, we assume that the samples are generated from a random process involving an unobserved latent variable $\mathbf{z}$. This generative process consists of (1) generating a value $\mathbf{z}^{(i)}$ from a prior distribution $p_\theta(\mathbf{z})$ (where $p_\theta(x) = p(x|\theta)$); and (2) generate a sample $x^{(i)}$ from a conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$.

Unfortunately, there are cases where the posterior is intractable to compute and where sample methods are not efficient. In the case of where the marginal likelihood $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})d\mathbf{z}$ is intractable, the posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is also intractable. Recall that the EM algorithm implies the computation of the expectation of the log-likelihood of the posterior. This is, the EM algorithm can not be used in this case. In the case of having large datasets, sampling methods such as Monte Carlo would require expensive sampling per data point which makes it also computationally infeasible.

The solution proposed to this in [10] is to use a function to approximate the posterior $q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$, which is denoted the probabilistic *encoder*. The proposed method is to optimize both the parameters θ and ϕ jointly using the (variational) lower bound on the marginal likelihood of data point i. The lower bound is given by:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})] \tag{13}$$

We refer to the original work ([10]) for the derivation of this lower bound.

A *variational autoencoder* (VAE) [10] is then a model composed of an encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and a decoder $p_\theta(\mathbf{x}|\mathbf{z})$ that are usually modelled using neural networks and are trained jointly to maximize the lower bound $\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$. Generally, it is assumed that the prior $p_\theta(\mathbf{z})$ is an isotropic Gaussian and, therefore, $p_\theta(\mathbf{x}|\mathbf{z})$ to be a multivariate Gaussian.

2.4.1 Reparametrization Trick

Notice in (13) that this requires generating samples from $q_\phi(\mathbf{z}|\mathbf{x})$. It is not possible to backpropagate through a sampling operation. Thus, the reparametrization trick is used, which consists of turning a continuous random variable $\mathbf{z}$ into a deterministic variable $\tilde{\mathbf{z}}$ by applying a deterministic and differentiable transformation. Formally, given a noise variable $\epsilon \sim p(\epsilon)$, we can write $\mathbf{z} = g_\phi(\mathbf{x}, \epsilon)$ where g_ϕ is a deterministic function parameterized by ϕ.

Recall that a Monte Carlo estimate of the expectation of a function $f(\mathbf{z})$ with respect to a distribution $q(\mathbf{z})$ can be written as:

$$\mathbb{E}_{q(\mathbf{z})}[f(\mathbf{z})] = \int f(\mathbf{z})q(\mathbf{z})d\mathbf{z} \approx \frac{1}{S}\sum_{s=1}^{S} f(\mathbf{z}^{(s)}) \tag{14}$$

where $\mathbf{z}^{(s)} \sim q(\mathbf{z})$. Using the reparametrization trick, the expectation of $q_\phi(\mathbf{z}|\mathbf{x})$ becomes:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] = \frac{1}{L}\sum_{l=1}^{L} f(g_\phi(\mathbf{x}, \epsilon^{(l)})) \tag{15}$$

Finally, by applying this to equation (13), we obtain the *Stochastic Gradient Variational Bayes* (SGVB):

$$\tilde{\mathcal{L}}(\theta, \phi; \mathbf{x}^{(i)}) = \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}) - \log q_\phi(\mathbf{z}^{(i,l)}|\mathbf{x}^{(i)}) \tag{16}$$

The most common approach is to assume that $p(\epsilon) = \mathcal{N}(0, I)$. Additionally, using $g_\phi(\mathbf{x}, \epsilon) = \mu + \sigma \odot \epsilon$, where μ and σ are the output of the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and $\odot$ denotes element-wise multiplication. By using these, we can sample (in a differentiable manner) from $q_\phi(\mathbf{z}|\mathbf{x})$ by sampling ϵ. Then it can be seen that the lower bound from (16) for sample $\mathbf{x}^{(i)}$ becomes:

$$\tilde{\mathcal{L}}(\theta, \phi; \mathbf{x}^{(i)}) \approx \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}) + \frac{1}{2}\sum_{j=1}^{J}\Big(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\Big) \tag{17}$$

, where $\mathbf{z}^{(i)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(i)}$. We refer to (Section 3 of [10]) for the reasoning behind this lower bound.

3 Bayesian Graph Neural Networks

In this section, definitions of the four Bayesian GNNs selected for this work are given, this is, Bayesian Graph Convolutional Neural Networks, Beta-Bernoulli Graph DropConnect, Graph Variational Auto-Encoders, and Graph structure Estimation Neural Networks.

3.1 Bayesian Graph Convolutional Neural Network

The *Bayesian Graph Convolutional Neural Network* (BGCN) [28] is a model proposed in 2019, that tries a novel approach to solve previous limitations of the existing Graph Convolutional Neural Networks, as the deterministic nature of the models and the overfitting present when few labels are available. The method makes the assumption of modelling the weights of the network as a random variable and uses the Monte Carlo Dropout introduced in Section 2.2.

In general terms, the network used is an assortative mixed membership stochastic block model with model learnable parameters $\lambda = \{\pi, \beta\}$. Further details of this network won't be discussed in this paper, as it is not the scope of the analysis. More information can be found in [28].

Applying the equation (9) to the BGCN results in:

$$p(\mathbf{Z}|\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs}) = \int p(\mathbf{Z}|\mathbf{X}, \mathbf{W}, \mathcal{G}_{obs})p(\mathbf{W}|\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs})p(\mathcal{G}|\lambda)p(\lambda|\mathcal{G}_{obs})d\mathbf{W}d\mathcal{G}d\lambda \tag{18}$$

The process follows the following steps:

1. V samples of model parameters are sampled from the posterior distribution $p(\lambda|\mathcal{G}_{obs})$
2. With the samples λ_v we approximate the distribution $p(\mathcal{G}|\lambda_v)$ and we sample VN_G random graphs $\mathcal{G}_{i,v}$ from the joint distribution $p(\lambda, \mathcal{G}|\mathcal{G}_{obs})$
3. We sample S weights $\mathbf{W}_{s,i,v}$ of the BGCN using the dropout approximation (MC dropout in our case) from the distribution $p(\mathbf{W}|\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs})$
4. Finally, we compute the posterior distribution of the variable of labels Z: $p(\mathbf{Z}|\mathbf{X}, \mathbf{W}, \mathcal{G}_{obs})$

After this computation, the equation (18) is reduced to the following approximation:

$$p(\mathbf{Z}|\mathbf{X}, \mathbf{Y}_{obs}, \mathcal{G}_{obs}) \approx \frac{1}{V}\sum_{v=1}^{V}\frac{1}{N_G S}\sum_{i=1}^{N_G}\sum_{s=1}^{S} p(\mathbf{Z}|\mathbf{X}, \mathbf{W}_{s,i,v}, \mathcal{G}_{i,v}) \tag{19}$$

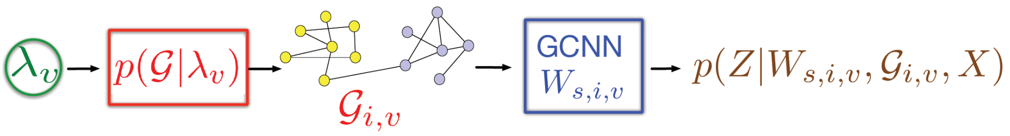
λ_v
$p(\mathcal{G}|\lambda_v)$
$\mathcal{G}_{i,v}$
GCNN
$W_{s,i,v}$
$p(Z|W_{s,i,v}, \mathcal{G}_{i,v}, X)$

Figure 2: Scheme of the functioning of a BGCN. Source: [28].

3.2 Beta-Bernoulli Graph DropConnect (BBGDC)

Beta-Bernoulli Graph DropConnect (BBGDC) [8] is a generalization of the stochastic regularization techniques presented in Section 2. BBGDC consists of allowing GNNs to draw different random masks for each channel and edge independently. Formally, a BBGDC layer is defined as:

$$\mathbf{H}^{(k+1)}[:,j] = \sigma\left(\sum_{i=1}^{f_k} \mathfrak{N}(\mathbf{A}\odot\mathbf{Z}_i^{(k)})\,\mathbf{H}^{(k)}[:,i]\,\mathbf{W}^{(k)}[i,j]\right) \quad \text{for } j=1,\dots,f_k \tag{20}$$

In the formula $\mathfrak{N}$ is the normalising operator, $\mathbf{H}^{(k)}$ is the k hidden layer and $\mathbf{W}^{(k)}$ is the k weight matrix, all these matrices have the same dimension as the $\mathbf{A}$. f_k is the number of features at layer k and $\mathbf{Z}_i^{(k)}$ is a sparse random matrix (with the same sparsity as the adjacent matrix $\mathbf{A}$) whose elements are non zero and drawn by a random $\text{Bernoulli}(\pi_k)$. This last parameter is the one that enables the network to adjust the binary mask for the edges, nodes and channels.

Compared to DropOut, DropEdge and Node Sampling, BBGDC has a free parameter $\mathbf{Z}_{i,j}^{(l)} \in \{0,1\}$ that can mask at the same time edges, nodes and channels. Note that the three regularization techniques mentioned are special cases of BBGDC.

3.2.1 GDC as Bayesian Approximation

In general terms, GDC applies a random masking to the adjacency matrix of the graph so to regularise the aggregation steps at each layer of the GNNs. 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 an appropriate Bayesian extension of GNNs.

$$h_v^{(k+1)} = \text{AGG}\left(\left\{\text{ACT}\left((\mathbf{z_{uv}}^{(k)}\odot h_u^{(k)})\mathbf{W}^{(k)} + b^{(k)}\right), u\in\mathcal{N}(v)\right\}\right) \tag{21}$$

$$= \text{AGG}\left(\left\{\text{ACT}\left(h_u^{(k)}(diag(\mathbf{z_{uv}}^{(k)})\mathbf{W}^{(k)}) + b^{(k)}\right), u\in\mathcal{N}(v)\right\}\right) \tag{22}$$

That by just setting $\mathbf{W}_{vu}^{(k)} := \mathbf{z}_{vu}^{(k)}\,\mathbf{W}^{(k)}$

$$h_v^{(k+1)} = \text{AGG}\left(\left\{\text{ACT}\left(\mathbf{W_{vu}}^{(k)} h_u^{(k)} + b^{(k)}\right), u\in\mathcal{N}(v)\right\}\right) \tag{23}$$

GDC can be seen as an approximating distribution $q_\theta(\boldsymbol{\omega})$ for the posterior $p(\boldsymbol{\omega}\,|\,\mathbf{A},\mathbf{X})$ 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 e-th 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 $\text{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}))$ 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.

3.2.2 Varitational inference Beta-Bernoulli GDC

The drop rate in BBGDC is usually set as a constant hyperparameter. A novelty introduced here by the authors is to learn the drop rate along with the other GNN parameters. This allows later to develop a variational inference algorithm for learnable BBGDC.

First note that all the all the drop masks are independent between them, hence for brevity the following expressions are only applied to one mask. The authors first propose a beta-Bernoulli prior to the binary random mask:

$$\begin{aligned} \hat{a}_e^{(k)} &= z_e^{(k)} a_e, \quad z_e^{(k)} \sim \text{Bernoulli}(\pi_k), \\ \pi_k &\sim \text{Beta}(c/K, c(K-1)/K), \end{aligned} \tag{24}$$

a_e is an element of the adjacency matrix corresponding to an edge e. $\hat{a}_e^{(k)}$ is this same element but masked with $z_e^{(k)}$.

To perform variational inference over BBGDC random masks, and the corresponding drop rate, the authors define the variational distribution as $q\mathbf{Z}^{(1)}, \pi_l = q(\mathbf{Z}^{(1)}, \pi_1 q(\pi_l)$, where $q(\pi_l)$ is a Kumaraswamy distribution. Here we omit the resulting KL divergence term and the approximation of gradients of the variational loss, however, we refer the reader to the original work, where the derivation shows an alternative to the reparametrization trick presented in 2 (notice that the discrete nature of the random masks, a reparametrization cannot be applied).

3.3 Variational Graph Auto-Encoders (VGAE)

Variational Graph Auto-Encoders (VGAE) [12] are a generalization of VAEs to graph structure. Recall that VAEs assumed that samples from a dataset where generated from a latent unobserved variable $\mathbf{z}$. In the case of VGAEs, it is assumed that the structure of the graph, given by the adjacency matrix $\mathbf{A}$, is generated from a latent variable $\mathbf{Z} \in N \times D$, where N is the number of nodes in the graph and D is the dimension of the latent space. The rest of the model is a generalization of the VAE model to graphs. On one side we will have the an encoder model $q_\phi(\mathbf{Z}|\mathbf{X}, \mathbf{A})$, which, as explained in Section 2 is an approximation to the posterior $p_\theta(\mathbf{Z}|\mathbf{X}, \mathbf{A})$, where $\mathbf{X} \in N \times F$ is the feature matrix of the nodes, with F being the number of features per node. On the other we have the generative process or decoder $p_\theta(\mathbf{A}|\mathbf{Z})$. Given this generative process, the probability of having an edge between nodes i and j is given by $p_\theta(\mathbf{A}_{ij} = 1|\mathbf{Z})$.

The formulation provided so far is the same one as in [12]. It can be seen that this formulation is for edge prediction tasks, where the goal is to predict the probability of having an edge between two nodes. Note however that the definition can be extended also to include the features of the nodes in the generative model, this is, $p_\theta(\mathbf{A}, \mathbf{X}|\mathbf{Z})$. Similarly, a featureless approach can be obtained by dropping the feature matrix $\mathbf{X}$ from the models.

In terms of training, VAEs are trained similarly to VAEs, this is, by optimizing the variational lower bound of (13). The resulting lower bound for VGAEs is given by:

$$\mathcal{L}(\theta, \phi; \mathbf{X}, \mathbf{A}) = \mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p_\theta(\mathbf{A}|\mathbf{Z})] - D_{KL}(q_\phi(\mathbf{Z}|\mathbf{X}, \mathbf{A})||p_\theta(\mathbf{Z})) \tag{25}$$

Normally, a Gaussian prior is assumed $p_\theta(\mathbf{Z}) = \prod p_\theta(\mathbf{z}_i) = \prod \mathcal{N}(\mathbf{z}_i|0, I)$, and the encoder is modelled as a Gaussian distribution $q_\phi(\mathbf{Z}|\mathbf{X}, \mathbf{A}) = \prod q_\phi(\mathbf{z}_i|\mathbf{X}, \mathbf{A}) = \prod \mathcal{N}(\mathbf{z}_i|\mu_i, \sigma_i^2)$, where μ_i and $\log \sigma_i^2$ are the output vectors of GNNs models.

Empirically, for edge prediction tasks, the generative model is usually modelled by a simple a dot product between the latent variables of the nodes, this is,

$$p_\theta(\mathbf{A}_{ij} = 1|\mathbf{Z}) = \sigma(\mathbf{z}_i^T \mathbf{z}_j) \tag{26}$$

where σ is the sigmoid function.

3.4 Graph structure Estimation Neural Networks (GEN)

Firstly let us define some notation, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V}$ is the set of N nodes, $\mathcal{E}$ is the set of edges, and $\mathbf{X} = [x_1, x_2, \cdots, x_N] \in \mathbb{R}^{NxD}$ represents the node feature matrix and x_i is the feature vector of node v_i.

In the semi-supervised node classification setting, a small part of nodes $\mathcal{V}_L = v_1, v_2, \cdots, v_l$ have labels corresponding to $\mathcal{Y}_L = y_1, y_2, \cdots, y_l$. Then, the goal of this model is to be able to simultaneously learn an optimal adjacency matrix $S \in \mathcal{S} = [0, 1]^{NxN}$ and GNN parameters Θ to improve node classification performance for unlabelled nodes.

In broad terms, this model uses an iterative three-step procedure to estimate the optimal graph and to optimise unlabelled classification. Firstly, it creates a set of observations $\mathcal{O} = \{\mathbf{A}, \mathbf{O}^{(0)}, \mathbf{O}^{(1)}, \cdots, \mathbf{O}^{(l)}\}$, where A is the original graph, and $\mathbf{O}^{(i)}$ is derived from a kNN strategy applied to the node representation $\mathbf{H}^{(i)}$ extracted with vanilla GCN layers with parameters Θ. Afterwards, these observations $\mathcal{O}$, predictions $\mathbf{Z}$ (that correspond to a row-wise softmax applied to the last layer $\mathbf{H}^{(l)}$), and labels $\mathcal{Y}_L$ are put into a graph estimator to infer the adjacency matrix $\mathbf{Q}$, where Q_{ij} denotes the probability that there is an edge between node v_i and node v_j. And lastly, extracting the next graph $\mathbf{S}$ from $\mathbf{Q}$ with a threshold ϵ. This process is repeated iteratively optimising the Θ parameter of the GCNs to better the community label estimation and optimising the structure estimation of the graph with the graph estimator.

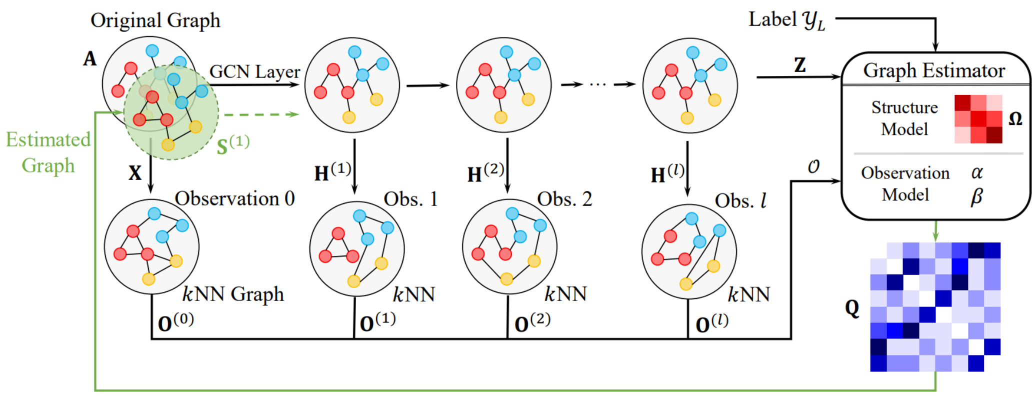

Figure 3: Scheme of the general structure of the Graph structure Estimation Neural network. Source: [24].

3.4.1 Observation Construction

To construct the observation set $\mathcal{O}$, we feed the original graph into vanilla GCNs. These observation sets help smooth out the structural information found in real-world complex systems, as they are often noisy. Specifically, we follow a neighbourhood aggregation strategy in each k^{th} layer of GCN as

$$\mathbf{H}^{(k)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}\right) \tag{27}$$

where $\tilde{\mathbf{A}}$ is the normalized adjacency matrix and $\tilde{D}_{ii} = \Sigma_j \tilde{A}_{ij}$. And lastly, $\mathbf{W}^{(k)}$ is a layer-wise trainable weight matrix, and σ denotes an activation function. As mentioned before, the $\mathbf{H}^{(k)}$ is the matrix of node representations of the k^{th} GCN layer. To simplify, we will also define the parameters of GCN as $\Theta = (\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(l)})$ that can be trained via gradient descent.

Subsequently, to extract the observations we fix the parameters Θ we take the node representations $\mathcal{H} = \{\mathbf{H}^{(0)}, \mathbf{H}^{(1)}, \cdots, \mathbf{H}^{(l)}\}$ to construct the observed kNN optimal graphs $\{\mathbf{O}^{(0)}, \mathbf{O}^{(1)}, \cdots, \mathbf{O}^{(l)}\}$, with $\mathbf{O}^{(i)}$ is the adjacency matrix of the corresponding $\mathbf{H}^{i}$. Since the original graph $\mathbf{A}$ is also an important observation we append it to the set of kNN graphs to get observation set $\mathcal{O}$.

Lastly, we feed the observation set $\mathcal{O}$, predictions $\mathbf{Z} = \mathbf{H}^{(k)}$ with a row-wise softmax layer on the l^{th} layer, and the labels $\mathcal{Y}_L$ to a graph estimator to infer the posterior distribution of the graph structure.

3.4.2 Graph Estimator

The observations obtained reveal optimal graph structure from different perspectives, but they may be unreliable or incomplete, given that we do not have a priori information on how accurate each one is we need to calculate the probability of the given structures. Bayesian inference allows us to reverse the probability of mapping a graph with a given community structure to the observations we found, and then compute the posterior distribution of the graph structure.

Firstly, let us denote $\mathbf{G}$ as the optimal graph symmetric adjacency matrix we are trying to estimate. We then apply the *Stochastic Block Model* (SBM) to compute the probability of $P(\mathbf{G}|\Omega, \mathbf{Z}, \mathcal{Y}_L)$, where Ω represents the parameters of the SBM which represents the probability of an edge between nodes depends only on their communities. For instance, the probability that a node v_j and v_i are connected depends on the communities of the nodes, c_i and c_j. Then, given the parameters of the probabilities, we can formalise the probability as

$$P(\mathbf{G}|\Omega, \mathbf{Z}, \mathcal{Y}_L) = \prod_{i<j} \Omega_{c_i c_j}^{G_{ij}} (1 - \Omega_{c_i c_j})^{1-G_{ij}} \tag{28}$$

where using the predictions $\mathbf{Z}$ we can define

$$c_i = \begin{cases} y_i & \text{if } v_i \in \mathcal{V}_L, \\ z_i & \text{otherwise} \end{cases} \tag{29}$$

This equation (28) represents our prior knowledge of the underlying structure before using the observations of the data. Therefore, we introduce an observational model to describe the relation between the optimal graph $\mathbf{G}$ and the observations $\mathcal{O}$. The observed edges are assumed that they follow an independent identically distributed Bernoulli distribution, conditional on the presence or absence of an edge in the optimal graph.

Then we can define the probability $P(\mathcal{O}|\mathbf{G}, \alpha, \beta)$ of the observations $\mathcal{O}$ given the optimal graph $\mathbf{G}$ and model parameters α and β. Particularly, α and β represent the true-positive and false-positive rate of observing an edge that exists or is not in the optimal graph $\mathbf{G}$. Finally, let us suppose that given the M observations (*i.e.*, $|\mathcal{O}|$), we observe an edge on E_{ij} of them, and no edge on the remaining $M - E_{ij}$. Then, using these definitions we can formally write

$$P(\mathcal{O}|\mathbf{G}, \alpha, \beta) = \prod_{i<j} [\alpha^{E_{ij}} (1-\alpha)^{M-E_{ij}}]^{G_{ij}} \times [\beta^{E_{ij}} (1-\beta)^{M-E_{ij}}]^{1-G_{ij}} \tag{30}$$

Lastly, using these models for structure and observation we can estimate the graph using Bayesian inference. Applying Bayes rule we have

$$P(\mathbf{G}, \Omega, \alpha, \beta|\mathcal{O}, \mathbf{Z}, \mathcal{Y}_L) = \frac{P(\mathcal{O}|\mathbf{G}, \alpha, \beta) P(\mathcal{G}|\Omega, \mathbf{Z}, \mathcal{Y}_L) P(\Omega) P(\alpha) P(\beta)}{P(\mathcal{O}, \mathbf{Z}, \mathcal{Y}_L)} \tag{31}$$

where $P(\Omega)$, $P(\alpha)$, $P(\beta)$ and $P(\mathcal{O}, \mathbf{Z}, \mathcal{Y}_L)$ are the probabilities of the parameters and available data, which are assumed to be independent.

We can then get an expression of the posterior probability for Ω, α and β, by summing all possible values of optimal graph $\mathbf{G}$ as follows

$$P(\Omega, \alpha, \beta|\mathcal{O}, \mathbf{Z}, \mathcal{Y}_L) = \sum_{\mathbf{G}} P(\mathbf{G}, \Omega, \alpha, \beta|\mathcal{O}, \mathbf{Z}, \mathcal{Y}_L) \tag{32}$$

And by maximising this posterior probability in reference to Ω, α and β parameters, we get a maximum-a-posteriori (MAP) estimates. With these values we finally calculate an estimated adjacency matrix $\mathbf{Q}$ for optimal graph $\mathbf{G}$

$$Q_{ij} = \sum_{\mathbf{G}} q(\mathbf{G}) G_{ij} \tag{33}$$

where value Q_{ij} is the posterior probability of an edge existing between v_i and v_j. We then threshold the $\mathbf{Q}$ matrix to obtain an $\mathbf{S}^t$ estimated adjacency matrix for the estimated graph.

To finalise, we iteratively optimise these GCN parameters Θ and estimated adjacency matrix $\mathbf{Q}$ with the use of the new estimated graph $\mathbf{S}^t$ as input of the sequence for τ iterations. Further information about the iterative optimization can be found in Wang, et. al. (2021) [24].

4 Results

This section compares the performance of the different Bayesian GNNs presented throughout the paper. To compare the performance of the models, dataset Cora from Planetoids datasets collection [26] was used. The Cora Dataset is a network consisting on 2708 nodes, that represent scientific publications, and 5429 links representing the citation network. Each paper can belong to one of the seven classes available. Each node in the network is described by a dictionary of 1433 unique words binary value that indicates if the word is in the publication or not. [26]. This dataset is usually applied to classification tasks, where the model tries to predict to which class belongs each node, taking into account the data from the node, the neighbourhood and the structure of the network.

For each model, the idea has been to replicate the results of the semi-supervised node classification task described above, which can be found in the original papers and, afterwards, to try to compare which model achieves a better accuracy. For the sake of comparison, we have included the accuracy results on the Cora dataset from four (non-bayesian) GNNs that are considered baseline models in the field, this is, GCNs, which were introduced in Section 2, Graph Attention Networks (GAT) [23], Graph Isomorphism Network (GIN) [25] and Residual Gated GNNs (GatedGCN) [3]. The results obtained with each model is shown in Table 5.

Regarding the BGCN model, described in section 3.1, a similar approach to the original paper has been followed. Using as reference the original code (check in the section above), we have trained the model using 5 and 10 training labels for each class, and then constructed the data partitions randomly or by a fixed split. Results have been obtained pondering the accuracy of 10 runs. The details of the original experiments can be found in [28]. The results of the BGCN are presented in the Table 1:

Table 1: Accuracy obtained for different configurations on the BGCN model.

		Data Partitions	
		Random	Fixed
Labels	5	72.6 ± 0.5	73.0 ± 0.7
	10	77.2 ± 0.2	77.2 ± 0.4

If we compare the results obtained with the ones in the original paper, we can see that our training is similar to the ones achieved in 50 runs by the authors. The tendency is the same: using 10 labels the accuracy is higher than the one achieved with 5 and for 5 labels, the accuracy is better if the data partition is fixed. Our accuracy for 5 labels is 2 points smaller than the original and for 10 labels we have obtained a better accuracy for the fixed data partition. Overall, one can see that the results are reproducible.

As for the BBGDC, we used the same setup that was described in the original paper. The number of epoch in the training were reduced due to the large computational time it took to train with the Cora dataset. The results for this model are lower in accuracy and higher in variance, in part due to the difference in training times between the two models. The results for our BBGDC are presented in the following table:

Table 2: Accuracy obtained for different splits on the BBGDC model for the Cora dataset.

	Data split		
	Train	Validation	Test
Mean	1.0	81.2	80.3
σ	0.0	0.8	1.1

Regarding the GVAE, recall from Section 3, that the formulation of the network is well suited for edge classification tasks, this is, to predict whether a link connecting two edges exists in a graph. Following the same approach as in the original work from [12], we used the Cora dataset and made random splits on the graph that were trained to predict the existence or absence of an edge in the split graph. The validation and test split consisted on 5% and 10% of the edges respectively.

The performance of the model was measured with the Area Under the Curve (AUC) metric, as well as the Average Precision (AP) of the edge prediction. The results are shown in Table 3. As a comparison, two methods that, at the time of publication of GVAE have been included for comparison: *Spectral Clustering* (SP) [22] and *Deep Walks* (RW) [18]. From the results, we can simply conclude that GVAE has a higher predictive performance compared the other models, which means that the model is learning meaningful latent representations. As mentioned in the original work, a limitation to this the usage of a Gaussian prior with a cross product decoder, as the latter pushes latent representations (i.e. embeddings) far from the zero-center.

Table 3: Performance of GVAE on the Cora dataset edge prediction task. Results with † were obtained from the original work.

	Cora	
Method	AUC	AP
SC [22]	$84.6 \pm 0.01^{\dagger}$	$88.5 \pm 0.00^{\dagger}$
RW [18]	$83.1 \pm 0.01^{\dagger}$	$85.0 \pm 0.00^{\dagger}$
GVAE [12]	91.4 ± 0.01	92.6 ± 0.01

Now, regarding the GEN model previously described, we used a similar experiment setup to the original paper [24] and the one previously described in this work, simplified into a dataset (Cora) with 20 labels per class. The results obtained were lower compared to the original study using the GEN model due to not tuning the hyperparameters further to get better results. These results are summarized in the following table:

Table 4: Accuracy obtained for different splits on the GEN model for the Cora dataset.

		Data split		
		Train	Validation	Test
Cora	**Mean**	1.0	82.80	81.60
	σ	0.0	0.6	0.6

Finally, we show in the table 5 all the results obtained with our models for node classification as well as some of the baselines found in the bibliography. One can see that all the results are very similar, not finding any significant differences between the models (all accuracies are between 75 and 83). The model that performs the worst is the BGCN, which could be justified if one takes into account that it is one of the most simple models tried. Surprisingly, the GCN model, which follows the same architecture as the BGCN but without applying the Bayesian theory obtains an accuracy above 80. Although this result would need a further research, reproducing them in similar conditions, for this specific dataset it could mean that sampling examples from the distributions involved in the equation (18) using Bayesian inference doesn't add any valuable knowledge to the simulation.

Other variations of the GCN, such as the GCN-BBGDC, can improve the accuracy of the BGCN model, but likewise don't overcome the GCN metric. The GEN model is the one that has been tested that achieves the best accuracy. However, the best result has still been found in the GAT model.

Table 5: Accuracy and standard deviation obtained with GNNs and Bayesian GNNs on Cora dataset, averaged using 10 random seeds. Results with † were obtained from the original contribution. The remaining results were reproduced by us.

Method	**Cora**
GCN [11]	$81.4 \pm 0.5^{\dagger}$
GAT [23]	$83.0 \pm 0.7^{\dagger}$
GIN [25]	78.5 ± 1.8
GatedGCN [3]	76.9 ± 1.0
BGCN [28]	75.0 ± 0.3
BGCN [28]	$76.5 \pm 1.5^{\dagger}$
GCN-BBGDC [8]	80.3 ± 1.1
GEN [24]	81.6 ± 0.6

5 Conclusions

Overall, the study of Graph Neural Networks allows us to understand one of the most widely used families of algorithms for handling graph-structured data in Machine Learning. Furthermore, one field of research inside the GNN are Bayesian Graph Neural Networks, which include the Bayesian theory inside the architecture of the model in order to achieve a better performance. In this paper, we have presented the theoretical background of GNN as well as some mathematical tools in order to study four different models of Bayesian Graph Neural Networks.

In the first place, we have introduced the BGCN, a model that samples parameters from the main equation of the GCN. Next, we present the BBGDC, which assumes that the dropout layers of the model are a Beta-Bernoulli distribution. Following that we have studied VGAE, which states that the graph nodes are generated by a latent unobserved variable and approximate its posterior distribution using NN; and finally we have described the GEN, which analyzes the optimization of the posterior distribution of the optimal observations of the graph from different perspectives.

After detailing all the model fundamentals, we have implemented each algorithm, using the original codes from the authors as a reference. The VGAE algorithm is designed to perform edge classification, whereas the other 3 are used for node classification purposes. In general terms, We have been able to reproduce the behaviour reported by the main papers on the topic using the Cora dataset. Furthermore, we have compared the performance of the models used for node classification with some other results found in the bibliography, showing that similar accuracies are achieved among all models and that, in a first approach, there is not a significant improvement when applying Bayesian concepts to GNN, although more extensive research should be done in order to reproduce the simulations in similar conditions.

In conclusion, the field of GNN is a state-of-the-art approach to work with data embedded in a graph that has been developed during the last decade, and Bayesian statistics allow us to explore new architectures of GNN in order to achieve a better performance in predictions.

References

[1] Cora dataset. Available from `https://graphsandnetworks.com/the-cora-dataset/`.

[2] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

[3] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

[4] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[5] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2016.

[6] Shengbo Gong, Jiajun Zhou, Chenxuan Xie, and Qi Xuan. Neighborhood homophily-based graph convolutional network. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23. ACM, October 2023.

[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[8] Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. Bayesian graph neural networks with adaptive connection sampling, 2020.

[9] Zhihua Jin, Yong Wang, Qianwen Wang, Yao Ming, Tengfei Ma, and Huamin Qu. Gnnlens: A visual analytics approach for prediction error diagnosis of graph neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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

[13] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[14] Maxime Labonne. Graph convolutional networks: Introduction to GNNs, 2023. Available from `https://towardsdatascience.com/graph-convolutional-networks-introduction-to-gnns-24b3f60d6c95`.

[15] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[16] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[17] Niklas Mäki. Bayesian graph neural networks : An empirical evaluation, 2023.

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

[19] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. The truly deep graph convolutional networks for node classification. *arXiv preprint arXiv:1907.10903*, 5, 2019.

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

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[22] Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data mining and knowledge discovery*, 23:447–478, 2011.

[23] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[24] Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie. Graph structure estimation neural networks. In *Proceedings of the Web Conference 2021*, pages 342–353, 2021.

[25] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[26] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[27] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33:17009–17021, 2020.

[28] Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Üstebay. Bayesian graph convolutional neural networks for semi-supervised classification, 2018.