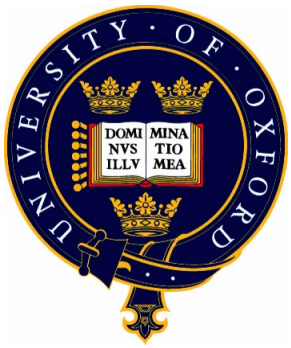

University of Oxford

Engineering Science - 4th Year Project

Bayesian Graph Structure Learning with applications to Musical Genre Classification

---

Orlando Fraser

May 2021

Project Supervisors:

Xiaowen Dong (Oxford-Man Institute of Quantitative Finance)

Pawan Kumar (Oxford OVAL Group & Deepmind)

Abstract

Geometric Deep Learning has recently emerged to become the paradigm of choice for applying machine learning to graph structured data. Within this field, a range of Graph Neural Network (GNN) architectures have emerged that have achieved state-of-the-art performance on a wide range of graph datasets. However, these approaches treat the graph provided as ground truth, having no way to deal with uncertainty in its structure created due to noise or modelling errors during the graph's formation. The field of Graph Structure Learning (GSL) was created in an attempt to deal with uncertain graphs through optimizing their structure as part of the learning process. Bayesian GSL algorithms are one research area within this field that we believe is particularly promising due to the recent successes of the Bayesian framework in a range of other machine learning tasks. Music genre classification is a well studied problem that has been shown to benefit from Geometric Deep Learning approaches and is therefore a relevant domain to test graph learning algorithms on. In this work, we survey the current literature on Bayesian GSL and apply it to the music genre classification problem. We discuss the differences in the empirical performance of the current Bayesian GSL algorithms and attempt to motivate these differences in the theoretical underpinnings of the models. Finally, we highlight the scalability issues with the current methods and propose modifications to the models to mitigate some of these issues.

Acknowledgements

Many thanks to Xiaowen and Pawan for a year of fun, insightful, and thought provoking conversations.

Contents

1 Introduction

Graphs are a ubiquitous data structure that can compactly represent a set of data points along with the relations between them. There are many situations where one would like to learn functions over graphs [1], and therefore machine learning on graph structured data is an area of very active research. In recent years, deep learning has successfully been able to leverage improvements in computer hardware to become the dominant paradigm for a wide range of machine learning tasks. However to utilize deep learning, one requires Euclidean data such as set of images or a time series, graphs are inherently non-Euclidean and therefore these algorithms cannot be applied. This motivated research into how to generalize deep learning to non-Euclidean domains, hence the field of Geometric Deep Learning was created [2].

Graph Structure Learning (GSL) [10] is a subfield of Geometric Deep Learning that aims to allow one to apply Geometric Deep Learning approaches in situations where the relational structure provided is imperfect with respect to a given function one aims to learn over it. Most current algorithms assume a ground truth graph that can be relied on absolutely during training and inference, however this is unlikely to be achieved in most real world situations due to modelling errors and noise being present during the formation of the graph structure. It is therefore very useful to have Geometric Deep Learning algorithms that do not make this strong assumption, allowing the effective application of graph learning algorithms on a wider range of data sources.

Despite the newness of the field, there have been various different directions taken within GSL. Bayesian approaches are one area which we believe is particularly promising for a number of reasons, most notably being the recent successes of the Bayesian framework in a wide range of other machine learning research areas [3].

Music genre classification is a popular machine learning problem that has been shown to benefit from Geometric Deep Learning approaches by Sathyamurthy et al. 2020 [4]. For a number of reasons, the graph structures used in [4] are not necessarily optimal with respect to the task and therefore we believe that GSL could be used to improve classification accuracy here. To our knowledge, no work has been completed applying GSL on this problem to date.

The goal of this work is to explore Bayesian approaches to GSL and apply them to the problem of music genre classification. The layout of the report is as follows:

In section 2, we introduce the machine learning problem of music genre classification and discuss the datasets used in this work. We also explain the motivations behind applying Graph Neural Networks

(GNNs), and more specifically GSL, to this problem.

In section 3 we present the motivations for Geometric Deep Learning and how it relates to the wider field of deep learning. We then provide the theory behind two popular GNN architectures used today; the Graph Convolutional Network (GCN) and the Graph Attention Network (GAT).

In section 4 we discuss the limitations of GNNs in situations of imperfect relational structure and how this motivates the research area of GSL. We then provide an overview of the current literature in the field and explain why we think Bayesian approaches should be the focus of research here. Finally, we present two current Bayesian models; the Variational Graph Convolutional Network (VGCN), and the Bayesian Graph Convolutional Network (BGCN).

In section 5, we highlight issues with the current Bayesian GSL approaches, most significantly being their scalability. We then provide a number of adjustments to the VGCN to mitigate some of these issues.

In section 6, we detail the experiments carried to test our algorithms on the music genre classification task. We compare and contrast results and attempt to provide theoretically motivated explanations for their differences. We also analyze the graphs learnt during the training of the models in an attempt to better understand the strengths and weaknesses of the different approaches.

Finally, in section 7 we conclude the work with a summary of contributions made. We also provide our views on current limitations and important future directions that should be taken in both GSL and music genre classification.

2 Music Genre Classification

In this section, we introduce the machine learning problem of music genre classification and discuss relevant work already completed in the area. We then provide our motivations behind applying GSL to this task. Finally, we introduce the datasets used in this work to test our algorithms.

2.1 Background

Musical genres are categories for songs that have arisen organically over time to characterize similarities in musical structure, allowing for the organization of musical collections. With the recent rise of digital streaming services for music, genre detection is becoming increasingly important for the automated tagging and recommendation of new music. Genres are a hierarchical categorization that can be represented as a series of tree data structures. The root of each tree is a root genre, and the children of this are the sub-genres of that root genre. We consider two multiclass classification problems; the single label classification problem of determining the dominant root genre of a song, and the multi-label classification problem of determining all the root and sub genres for each song (note that multiple root genres are possible for a single song).

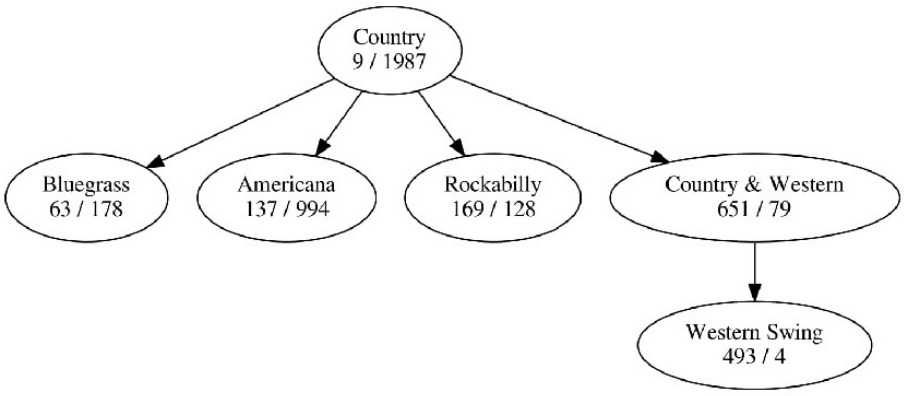

Figure 1: The genre hierarchy tree corresponding to the Country root genre. Left number is the genre ID in the FMA dataset and the right number is the number of tracks that contain the genre. Code from [21] is used to generate the plot.

2.2 Previous Work

The three main sources of information that are relevant for genre classification are the spectrogram of the song audio, the song review text, and the album cover image. Oramas et al. (2018) [20] demonstrated a multimodal approach that combined all three of the data sources by using neural networks to embed

them into a shared vector space. These representations can then be combined and fed into another neural network to perform the final classification of the song. The results not only achieved state-of-the-art classification accuracy, but also demonstrated that the combination of learned data representations from different modalities yields better results than any modality in isolation.

Sathyamurthy et al. 2020 [4] built on this work by incorporating Spotify related artist information in addition to the previous three sources. Spotify stores related artists for each artist and has an API that can be queried for any song present [5]. Specifically, for each artist, 20 related artists are found from the listening patterns of the 200 million users of the platform (note that the exact algorithm used here is proprietary knowledge that Spotify does not disclose publicly). This related artist data should be informative to the task of genre classification because of the prior belief that users tend to listen to music from a small set of genres and therefore songs by related artists are likely to share the same genre. We can therefore leverage this information to increase the accuracy of genre classification.

The challenge with this related artist data is that there is no simple way to represent it in a Euclidean fashion similar to what was done for the audio, text, and images. This is because the related artist data is inherently non-Euclidean. A clear way to represent this data is through a graph, where each song was a node and an edge existed between two songs if the artist of either one was present in the related artist set of the other. Additionally, the node features of the graph were given by the multimodal embeddings of the audio, text and image data. A GCN was then applied to resulting graph to classify the genres. This approach gave a significant boost in accuracy over [20], showing that incorporating non-Euclidean information was beneficial to the task.

2.3 Graph Structure Learning for Music Genre Classification

The Spotify related artist graph is a relational structure that has been shown to be helpful to the genre classification task. However, the question is whether or not this is the *optimal* relational structure with respect to the task?

Spotify stores a maximum of 20 related artists per artist, which means that the graph structure extracted is unlikely to fully describe all the artist relations in the data. This will therefore limit the information content of the graph produced and as a result will limit the expressivity of any function on the graph that one expects to learn. Additionally, related artists are not necessarily the only source of non-Euclidean structure that exists in the data. For these reasons, it seems unlikely that the Spotify related artist graph is the maximally informative one with respect to our task. GSL could be used to

improve our graph structure over what was provided by Spotify related artists, leading to an increase in classification accuracy. For these reasons, we think that the problem of genre classification would be a good area to test the effectiveness of our algorithms, contributing to the literature of both fields in one piece of work.

2.4 The FMA dataset with the Spotify graph

The Free Music Archive (FMA) Defferrard et al. 2017 [21] is a popular dataset for research on a range of machine learning tasks in music, genre classification being one of them. It provides 106,574 tracks from 16,341 artists arranged into 161 genres. Raw audio, album images, and text for each track is provided in addition to pre-computed feature embeddings from the three modalities. Higher accuracies should be able to be achieved via an end-to-end approach where a classifier is learnt directly from the raw data since some information will be lost through the pre-embedding process. However in this work we focus on the pre-computed embeddings only since the goal is to analyze the effectiveness of GSL algorithms on the problem rather than simply attempting to maximize the classification accuracy on the dataset.

Figure 3: Visualizations of the Cora (left) and Citeseer (right) graphs.

Figure 2 shows a visualization of the Spotify related artist graph applied to the FMA medium subset. We observe moderate clustering of nodes with the same root genre which indicates that our related artist graph structure is informative to some extent with respect to the genre classification task. A common pattern in the structure of clusters is one parent node connected to a large number of child nodes. This

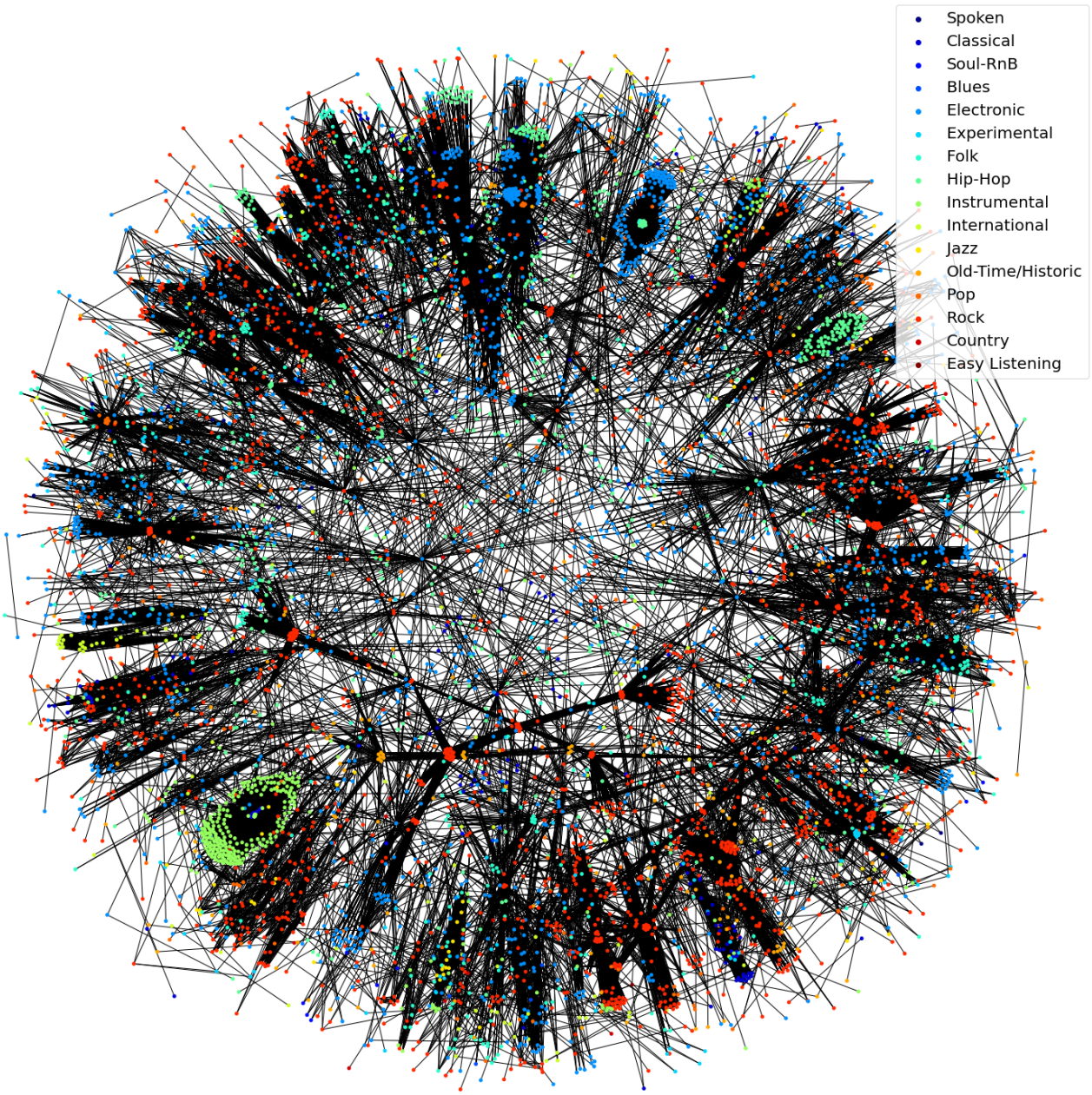

Figure 2: A visualization of the graph generated from the Spotify related artist data on the FMA medium subset. Nodes colours are assigned based on the root genre of the corresponding song. A force-directed graph drawing algorithm [22] was used to plot the graph in a way that simultaneously minimizes edge crossings and edge lengths. This will have the affect of positioning densely connected sets of nodes near to each other, therefore allowing us to optimally visualize the community structure in the dataset.

structure is not ideal because the child nodes will be forced to rely very heavily on a single parent node which will make any classifier learnt on the data more likely to overfit. This is because any noise or other errors than exist in these parent nodes will be amplified significantly, leading to weaker generalization performance. Additionally, one can see a large number of edges that cross between label communities. This reduces the community structure of the graph, making it less informative for the classification task.

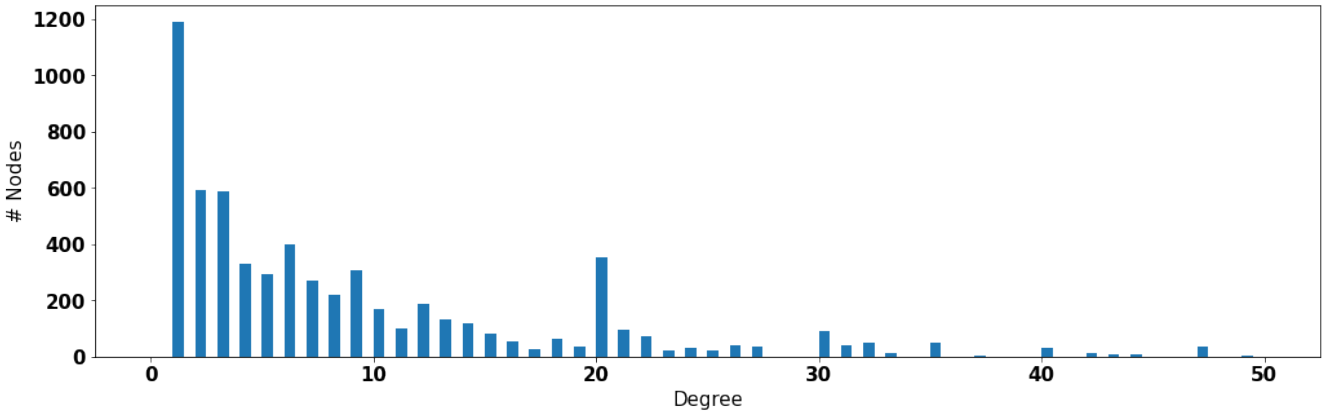

Figure 4: The degree distribution for the Spotify graph.

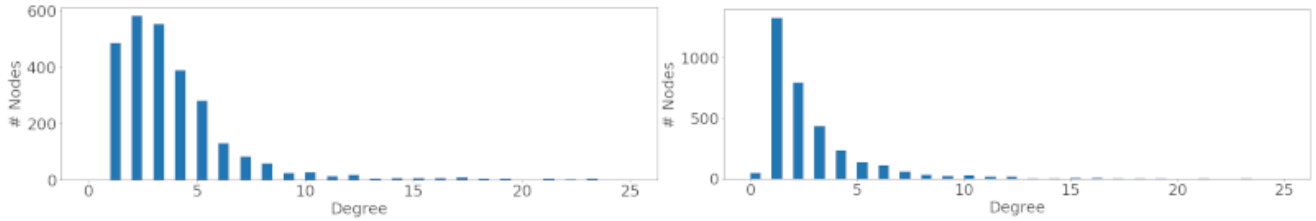

Figure 5: Degree distributions for Cora (left) and Citeseer (right) graphs.

To better understand what a good graph structure might look like, which analyze the Cora [23] and Citeseer [24] citation network datasets. Here, nodes are academic papers and an edge exists between two nodes if one paper cites the other. The nodes are labelled and therefore they are popular benchmark graphs for testing semi-supervised node classification algorithms. Visualizations of these graphs and degree distributions are provided in Figuress 3 and 5. We use them as more optimal graph structures that we can qualitatively and quantitatively compare and contrast against the Spotify graph. We can clearly see much stronger clustering of the graphs into their label groups which indicates that the relational structure is highly informative to the classification task and also that significantly less noise is present.

Figure 3 shows the degree distribution of the Spotify graph. We note the long tailed nature of the distribution which is caused by parent nodes that have very large number of edges whilst the majority have very few. We also note the irregularity at a degree of 20 that is caused by Spotify only storing a maximum

of 20 artists per artist. In comparison, Cora and Citeseer have much smoother degree distributions with a faster drop off at high degrees.

A graph's modularity is one measure of the extent to which community structure exists in a graph Newman 2006 [25]. The value of the modularity for unweighted and undirected graphs lies in the interval $Q \in [1/2, 1]$. It is positive if the number of edges within groups exceeds the number expected on the basis of chance. We found modularities of 0.96 and 0.95 for the medium and large subsets respectively, indicating strong community structure in the data. This does not however necessarily mean that the classification task will be straightforward as the communities found are not necessarily aligned with the node labels. For Cora and Citeseer, values of 0.79 and 0.88 were found. Visually, Cora and Citeseer appear to be more strongly clustered than the Spotify graph despite the lower modularities. This suggests than the communities found were more aligned with the labels and therefore the graph structure in these graphs would be more informative to the classification task.

In summary, this qualitative analysis has exposed the information content stored in the Spotify related artist data but has also revealed the weaknesses in the graph structure that arise from dataset limitations and noise. Real world graphs in a wide range of areas are much more likely to look like the Spotify graph than Cora and Citeseer, which highlights the need for more robust graph algorithms that can deal with imperfect input data. The Bayesian graph structure learning algorithms discussed in this paper are one approach to providing this robustness.

3 Graph Neural Networks

3.1 Graph Theory

A simple graph is given by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with nodes $v \in \mathcal{V}$ and edges defined by the triple $(u,\ v,\ a) \in \mathcal{E}$. Where u and v are the nodes at either end of the edge, and a is the edge weight. We can then define an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ that stores the edge weights. In the case of undirected unweighted graphs, A will be a symmetric boolean matrix. To simplify the notation, we denote the number of nodes in a graph $\mid \mathcal{V} \mid$, as N.

The degree matrix stores the number of edges incident to each node, it is given by $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. Furthermore, the graphs that we will be considering have a feature vector $\mathbf{x}_u \in \mathbb{R}^d$ associated with each node.

3.2 The Graph Neural Network Model

Our goal is to learn functions that embed the nodes of our graph in a vector space $f : \mathcal{V} \to \mathbb{R}^k$. In the case of node classification, $\mathbb{R}^k$ would be the logit space of a k class problem, where each element of the vector holds the logit for the corresponding class. The probabilities of each class can then be found by applying a softmax to the logits.

Standard embedding methods use a shallow approach to generate these node representations, where only the feature vector of a node is used to form the embedding for that node. An example of such a model would be an Multi Layer Perceptron (MLP). These methods ignore the relational structure present in the data and therefore they miss out on potentially useful information (if the relational structure provided is in fact relevant to the task). The motivation behind Graph Neural Networks (GNNs) is to make use of this relational structure, in addition to the feature vectors, in order to learn richer and more expressive representations over graphs.

Convolutional neural networks (CNNs) are a popular class of models that can learn functions on Euclidean data. Most commonly being images, where each pixel in the image is a single data point in the Euclidean space. CNNs work by encoding an image into a set of features that are relevant to the desired mapping by performing a series of hierarchical filtering operations on the input. These features can then be decoded to perform classification or regression of the input. The convolution operation acts as a local aggregator, allowing features to be extracted from the input that incorporate information from a neighborhood of pixels, rather than from just isolated pixels alone. The local aggregation operation is

essential for allowing functions to be learnt on graphs that utilize relational structure in the data, and therefore CNNs are an obvious place to draw inspiration from for functions on graphs.

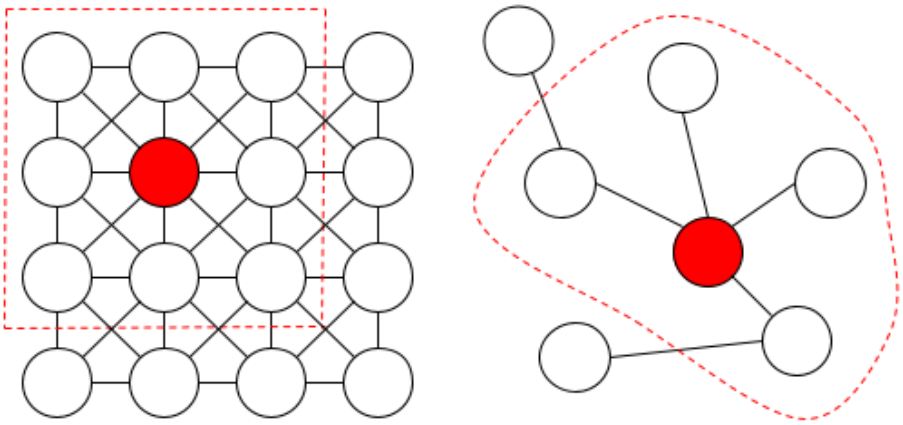

Figure 6: One can consider a 2D euclidean structure as a graph with a uniform relational structure (Left), where each node is connected to its 8 immediate neighbors only. This give a single canonical ordering of the nodes which means that functions over the graph can be designed to be applicable to this ordering only. Local aggregation of the node's immediate neighborhood can be applied via a convolution operation with a 3x3 filter, where each filter value is a learnable parameter. In general however, graphs do not have a uniform relational structure (right). There is no canonical ordering of nodes which means that functions over general graphs must be permutation invariant. The Euclidean convolution operator is not permutation invariant and therefore other local aggregation operators must be explored in order for functions to be learnt on graphs.

The key requirement for the applicability of a CNN to a data input is that the input data is Euclidean, ie it lies on a uniform grid. Graphs are an inherently non-Euclidean domain and so there exists no obvious way to apply CNNs to them. The motivation behind GNNs is to generalize the convolution operator from the CNN so that hierachical neighborhood aggregation can be applied on graphs.

The Neural Message Passing framework has been proposed where successive embeddings for each node are found through aggregation and update operations on the previous set of embeddings [8]. Multiple layers can be applied in series, producing deep hierarchical representations that allow highly non-linear functions over the graph to be learnt. The aggregation operation occurs over the neighborhood $\mathcal{N}(u)$ of the node u. Importantly, it must be a permutation invariant function as no canonical ordering of nodes exists. The output of the aggregator is combined with the node feature vector itself in the update function to produce the new representation. The layer update rule is defined as:

$$\mathbf{h}_u^{(k+1)} = U^{(k)}\left(\mathbf{h}_u^{(k)},\ A^{(k)}\left(\{\mathbf{h}_v^{(k)}\ \forall v \in \mathcal{N}(u)\}\right)\right) \tag{1}$$

$U^{(k)}$ and $A^{(k)}$ are general update and aggregation functions for layer k respectively. The embeddings for layer 0 are set to the node feature vectors provided, $\mathbf{h}_u^{(0)} = \mathbf{x}_u,\ \forall u \in \mathcal{V}$, and the output of the network

is set to the embeddings of the final layer $\mathbf{z}_u = \mathbf{h}_u^{(K)},\ \forall u \in \mathcal{V}$. Node embeddings at the output of the kth layer have information from the k-hop neighborhood of the node. Therefore, increasing the number of layers, has the affect of increasing the receptive field of the network. We now consider two popular instantiations of this general model.

3.3 Graph Convolutional Network

The Graph Convolutional Network (GCN) Kipf et al. 2016 [6] applied a localized first-order approximation of spectral graph convolutions. The key insight here was that a permutation invariant aggregation could be achieved if the convolution was applied in the spectral domain. In practice, this amounts to omitting the explicit update step and performing the aggregation step over the node itself as well as its neighbors. The approach has been shown to reduce the tendancy of the model to overfit, but at the expense of limiting expressivity as there is no way for the aggregator to differentiate information coming from the node neighbors and the node itself. The aggregator uses a weighted sum over nodes where each weight is found by normalizing with respect to the degree of the node. This ensures that information is aggregated somewhat evenly from all nodes, a feature that is especially useful in graphs with a high variance degree distribution.

$$\mathbf{h}_u^{(k+1)} = \sigma\left(\Theta^{(k)} \frac{1}{\sqrt{|\mathcal{N}(u)|}} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\mathbf{h}_v^{(k)}}{\sqrt{|\mathcal{N}(v)|}}\right) \tag{2}$$

$\sigma(\mathbf{x}) = max(0, \mathbf{x})$ is the element wise ReLU activation function and $\Theta^{(k)}$ is the learnable weight matrix for layer k. This node wise update rule can be vectorized to give the following graph level update rule.

$$\mathbf{H}^{(k+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(k)} \Theta^{(k)}\right) \tag{3}$$

Where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with added self connections, and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$ is the degree matrix of the graph with the self connections added. The GCN has proven to be one of the most effective and popular architectures, seeing widespread use in a range of research areas including computer vision and drug discovery [7].

3.4 Graph Attention Network

The Graph Attention Network (GAT) Veličković et al. 2017 [9] extends the convolution operator in the GCN by adding *attention*. This was shown to increase the expressivity of the neighborhood aggregation

by allowing the relative importance of node neighbors to be learnt. The idea is to assign a weight $\alpha_{u,v}$ that quantifies the importance of neighbor v to node u. The layer update rule is then given by:

$$\mathbf{h}_u^{(k+1)} = \sigma\left(\sum_{v \in \mathcal{N}(u) \cup \{u\}} \alpha_{u,v} \Theta^{(k)} \mathbf{h}_v^{(k)}\right) \tag{4}$$

The following attention model was proposed in the original paper, where $\oplus$ is the concatenation operation, and the matrix $\mathbf{W}$ and the vector $\mathbf{a}$ are further learnable parameter arrays.

$$\alpha_{u,v} = \frac{\exp(\mathbf{a}^T[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v])}{\sum_{v' \in \mathcal{N}(u)} \exp(\mathbf{a}^T[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}])} \tag{5}$$

To increase modelling capacity, Multi-head attention is used. C different attention mechanisms (given by equation 4) are computed and then the results are concatenated to form the embedding of that node for that layer.

The motivation behind the GAT model is the prior belief that some nodes are more informative than others with respect to some function over the graph. The attention mechanism provides a way to differentiate the important nodes in a flexible and learnable manner. In music genre classification on a song graph, less informative songs would be ones that have an ambiguous genre, for example. The GAT model was able to slightly outperform the GCN on both Cora and Citeseer by a couple of percentage points, indicating the strengths of the attention model in this context.

4 Graph Structure Learning

The GNNs discussed so far all require data that has a clearly defined graph associated. However situations may occur where no graph is provided or the graph provided is noisy/incomplete, but we still would like to use graph models on the data. This situation would occur if we have a prior belief that some non-Euclidean relational structure exists between data points that we believe will be informative to the function over the data points that we are trying to learn. Applying a model designed for Euclidean data, such an MLP, will therefore limit the expressivity of a function learnt on the data since all of the non-euclidean structure would be ignored. The goal of Graph Structure Learning (GSL) is to learn a maximally informative graph structure with respect to a given task. This learnt graph structure can then be used with graph learning algorithms (such as the ones discussed previously) to perform inference on the data. Additionally the learnt graph can be an end point itself, potentially conveying important structural insights about the data such as community structure.

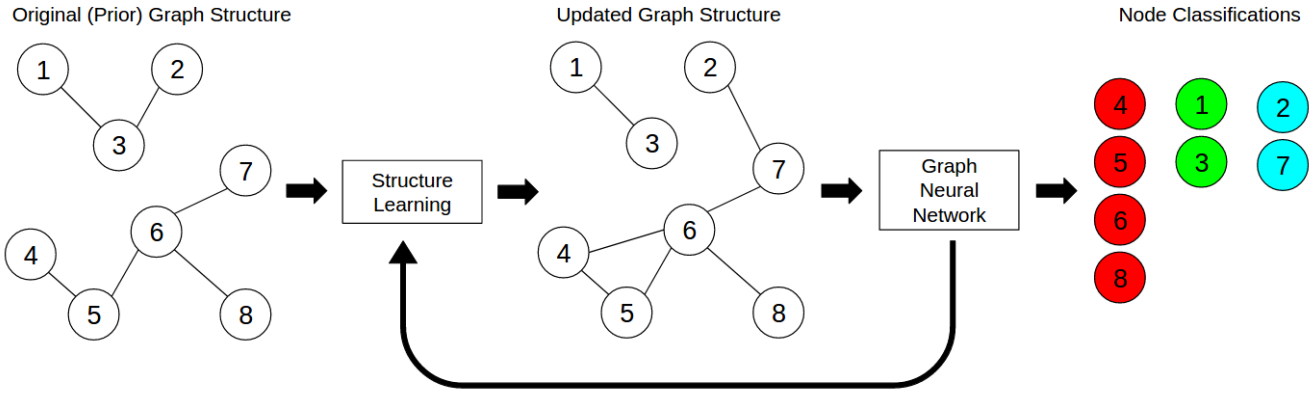

Figure 7: The general formulation of GSL for node classification. The original graph is successively updated via a structure learning algorithm and then fed into a GNN model to find the embedding for each node. In some models, feedback is used and the process is repeated until desired properties of the graph structure are achieved.

The current literature on GSL is fairly minimal as the research area is only a couple of years old. The general formulation is to alternatively or jointly optimize both a structure learning function and a node embedding function. As we have discussed previously, GNN architectures can achieve state-of-the-art performance on the node embeddings and are therefore consistently used for the embedding stage. On the other hand, there has been no consensus on the class of structure learning algorithms to use. There are three main directions that have been taken so far; metric learning, probabilistic modelling, and direct optimization [10]. We provide a brief overview of them here.

In Metric Learning, a kernel function is learnt that maps two nodes to the weight of the edge connecting

them [12]. $\tilde{\mathbf{A}}_{ij} = \phi(u, v)$, where intermediate node embeddings are used as input. Popular choices for the kernel function ϕ are a gaussian kernel or a cosine similarity. The mapping is performed for all node pairs, allowing a graph to be constructed. One issue with this approach is that its difficult to incorporate information from the prior graph into the structure learning algorithm. This means that the learned structure can vary wildly from the original, potentially leading to overfitting and other generalization issues.

Direct optimization approaches, treat the graph adjacency matrix as a learnable matrix of parameters. One can construct an optimization objective that incorporates graph priors as well as the accuracy of the GNN embedding. This can then be jointly optimized over both the adjacency matrix and the GNN parameters.

Finally, Probabilistic approaches have been applied where the graph structure is modelled as a realization from a generative model. One can impose structural constraints on the generative model which enforces certain properties to be present in the sampled graphs. For example, NeuralSparse Zheng et al. 2020 [11] defines a model which generates a k nearest neighbor graph. This flexibility allows prior beliefs to be incorporated strongly into the graph generation process.

4.1 Bayesian Graph Neural Networks

Bayesian models take the probabilistic approaches one step further by not relying on a single graph structure, but instead marginalizing over all possible realizations of the generative model. The motivation behind this approach is to be able to account for uncertainty and noise in the underlying graph structure by not relying on any single graph as a ground truth representation of the relations between nodes. Recent successes of Bayesian Neural Networks in various other areas of machine learning [3] have shown the real strengths of this framework for dealing with uncertainty and therefore we think that it should be the primary direction of research within GSL. The focus of this report is to compare and contrast the current Bayesian GSL methods and test them on the task of music genre classification.

Treating the data and the model in a fully Bayesian fashion, the posterior distribution for the labels conditioned on all of the observed data is given by the integral in equation (6). Note that the graph is fully defined by its adjacency matrix and so a distribution over a graph is equivalent to a distribution over the graphs adjacency matrix $\mathbf{A}$. We consider our original graph as an observation from the generative model and denote it by the matrix $\mathbf{A}_{obs}$.

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{Y}_{obs}, \mathbf{A}_{obs}) = \int p(\mathbf{Y}|\mathbf{X}, \Theta, \mathbf{A}) p(\Theta|\mathbf{Y}_{obs}, \mathbf{X}, \mathbf{A}) p(\mathbf{A}|\mathbf{A}_{obs}, \mathbf{X}, \mathbf{Y}_{obs}) d\Theta d\mathbf{A} \tag{6}$$

Where $p(\mathbf{Y}|\mathbf{X}, \Theta, \mathbf{A})$ is the GNN based likelihood, $p(\Theta|\mathbf{Y}_{obs}, \mathbf{X}, \mathbf{A})$ is the GNN parameters posterior, and $p(\mathbf{A}|\mathbf{A}_{obs}, \mathbf{X}, \mathbf{Y}_{obs})$ is the posterior for the graph (the generative model). This integral is intractable and therefore must be approximated. A Monte Carlo approximation is given by:

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{Y}_{obs}, \mathbf{A}_{obs}) \approx \frac{1}{N_\Theta N_\mathbf{A}} \sum_{i=1}^{N_\Theta} \sum_{j=1}^{N_\mathbf{A}} p(\mathbf{Y}|\mathbf{X}, \Theta_i, \mathbf{A}_j) \tag{7}$$

Where Θ_i and $\mathbf{A}_j$ are samples from their respective posterior distributions. This approximation is an unbiased estimator of the true posterior which becomes more accurate the greater the number of samples that you take. It was shown by Gal and Ghahramani (2016) [13] that with a suitable choice of variational approximation to the posterior of Θ, one can draw samples Θ_i via Monte Carlo dropout. This is equivalent to applying dropout to the network parameters during inference as well as during training. The GNN forward pass is then stochastic, which allows uncertainty estimates over the label posterior with respect to the network parameters to be found. From here, the Bayesian approaches diverge, as there are a number of different directions that have been taken for finding a posterior distribution over the graph. We present two of these approaches next; a variational approximation, and an Assortative Mixed Membership Stochastic Block Model (MMSBM).

4.2 Bayesian Graph Convolutional Network

The Bayesian Graph Convolutional Network (BGCN) Zhang et al. 2018 [16] was the first Bayesian approach to graph structure learning that was proposed. An Assortative Mixed Membership Stochastic Block Model (MMSBM) is used to model the generative process of the graph. Our prior belief is that closely connected nodes in the graph are likely to belong to the same category and therefore a graph structure that has communities which align with the categories of our classification task should be beneficial to the function that we hope to learn. For this reason, we believe that an MMSBM is an appropriate generative model for the graphs we are considering.

Each node u is assigned a C dimensional community membership probability distribution $\pi_u = [\pi_{u1}, ..., \pi_{uC}]$ that specifies the probability of u belonging to each of the C communities. We also have a community strength parameter for each community $\beta_c \in [0, 1]$ that controls the probability of an edge between any pair of nodes within a community. Finally, there is a cross community link probability δ, for

simplicity this is set as a hyperparameter rather than being inferred during training. We use a maximum a posteriori estimate for π and β:

$$\{\hat{\pi}, \hat{\beta}\} = \underset{\beta,\pi}{\operatorname{argmax}}\, p(\beta, \pi|\mathbf{A}_{obs}) = \underset{\beta,\pi}{\operatorname{argmax}}\, p(\beta)p(\pi)p(\mathbf{A}_{obs}|\beta, \pi) \tag{8}$$

Beta(ν) and a Dirichlet(ζ) distributions are used for the prior on π_u and β_c respectively, where ν and ζ are hyperparameters. Stochastic gradient ascent is used to find the maxima of the posterior. Note that the generative model is only conditioned on the observed graph and not all of the observed data (node features and the observed labels). This means that the learnt graph structure is not dependent on any of the node features, we will discuss the ramifications of this in more detail in section 4.4.

Sampling a graph from the generative model is done as follows: For any two nodes u and v, Sample a community for each node $c_{uv} \sim \text{Cat}(\hat{\pi}_u)$ and $c_{vu} \sim \text{Cat}(\hat{\pi}_v)$. If $c_{uv} = c_{vu} = c$, then sample the edge as $a_{uv} = \text{Bernoulli}(\hat{\beta}_c)$, otherwise sample $a_{uv} = \text{Bernoulli}(\delta)$. Posterior inference of the labels can be carried out by sampling $N_{\mathbf{A}}$ graphs and then computing the Monte Carlo sum given by equation 7.

Note that in the original BGCN paper, the MMSBM optimization was computed within the loop over the graph samples, whereas here we fully optimize the MMSBM before taking any graph samples (step 1 in the algorithm). The motivation for this was that the training procedures for the MMSBM and the GNN are decoupled and therefore it made little sense to use sub-optimal values of π, β to draw graph samples for training the GNN. This hypothesis was experimentally validated.

Algorithm 1 BGCN

Input: $\mathbf{X}$, $\mathbf{Y}_{obs}$, $\mathbf{A}_{obs}$,
Output: p($\mathbf{Y}$ | $\mathbf{X}$,$\mathbf{Y}_{obs}$, $\mathbf{A}_{obs}$)

1: Find MAP estimate $\hat{\pi}, \hat{\beta}$ of the MMSBM parameters via stochastic gradient ascent of $p(\beta, \pi|\mathbf{A}_{obs})$
2: **for** i = 1 : $\mathbf{N}_{\text{A}}$ **do**
3: Sample graph $\mathbf{A}_i \sim p(\mathbf{A} \mid \hat{\pi}, \hat{\beta})$
4: Train GNN using graph A_i
5: **for** j = 1 : $\mathbf{N}_{\Theta}$ **do**
6: Sample weights Θ_{ij} via MC dropout of the GNN weight matrix $\hat{\Theta}$
7: Perform forward pass through the GNN using $\mathbf{A}_i$ and Θ_{ij}
8: Approximate p($\mathbf{Y}$ | $\mathbf{X}$,$\mathbf{Y}_{obs}$, $\mathbf{A}_{obs}$) by taking the average of all GNN outputs from step 7.

4.3 Variational Graph Convolutional Neural Network

The Variational Graph Convolutional Neural Network (VGCN) Elinas et al. (2020) [14] uses a variational approximation to the intractable graph posterior. To optimize the model, gradient based maximization

of the evidence lower bound (ELBO) is carried out with respect to both the variational parameters Φ and the GNN parameters Θ.

$$\mathcal{L}_{ELBO}(\Phi, \Theta) = \mathbb{E}_{q_\Phi(\mathbf{A})} \log p_\Theta(\mathbf{Y}_{obs}|\mathbf{X}, \mathbf{A}) - \gamma \cdot \mathrm{KL}(q_\Phi(\mathbf{A}) \parallel p(\mathbf{A})) \tag{9}$$

Where the first term is the expected value of the log likelihood for the observed labels and the 2nd term is the Kullback–Leibler (KL) divergence between the variational posterior and the prior for the graph. This KL term acts as a regularizer for the model by preventing the variational posterior from drifting too far from the prior. The relative strength of this regularizer is controlled with a damping factor $\gamma \in [0, 1]$.

We assume that the observed labels are conditionally independent, allowing us to write our overall likelihood as the product of the individual posterior label distributions for each node. These are simply given as a categorical distributions parameterized by the output of our GNN for each node.

$$p_\Theta(\mathbf{Y}_{obs}|\mathbf{X}, \mathbf{A}) = \prod_{y_i \in \mathbf{Y}_{obs}} p_\Theta(\mathbf{y}_i|\mathbf{X}, \mathbf{A}), \quad \text{where} \quad p_\Theta(\mathbf{y}_i|\mathbf{X}, \mathbf{A}) = \mathrm{Cat}(\mathbf{y}_i|\mathbf{z}_i) \tag{10}$$

The model requires two design choices. The first is the choice of graph prior. The authors in [14] use an independent Bernoulli distribution given by:

$$p(\mathbf{A}) = \prod_{ij} p(\mathbf{A}_{ij}) \quad \text{where} \quad p(\mathbf{A}_{ij}) = \mathrm{Bernoulli}(\mathbf{A}_{ij}|\rho_{ij}) \tag{11}$$

The Bernoulli parameters are found from a smoothed version of the observed graph $\rho_{ij} = (\mathbf{A}_s)_{ij}$ where $\mathbf{A}_s = \alpha\mathbf{A}_{obs} + \beta(1 - \mathbf{A}_{obs})$. Here, $\mathbf{A}_{obs}$ is a sparse binary adjacency matrix and then $(\mathbf{A}_s)_{ij} \in \{\alpha, \beta\}$ is a dense adjacency matrix which stores our prior uncertainty on the likelihood of an edge at any position in the graph. The hyperparameters α and β encode ones degree of belief on the likelihood of edge based on whether an edge existed in the observed graph, they can be varied to reflect changes in our prior beliefs here. No structural information is included in the prior, (eg clustering structure or sparseness) which may limit its ability to correctly model all of our prior beliefs on $\mathbf{A}$.

The other design choice is the functional form of the variational posterior. The authors in [14] focused on a free parameterization, where the set of variational parameters are $\Phi = \{\phi_{ij}\}$.

$$q_\Phi(\mathbf{A}) = \prod_{ij} p(\mathbf{A}_{ij}) \quad \text{where} \quad q_\Phi(\mathbf{A}_{ij}) = \mathrm{Bernoulli}(\mathbf{A}_{ij}|\phi_{ij}) \tag{12}$$

Unweighted graphs are discrete structures and therefore they must be modelled by a discrete distribution such as the one defined above. However sampling from a discrete distribution is a non-differentiable operation, which prevents the use of gradient based optimization algorithms that are required to maximize the ELBO. The authors in [14] therefore propose using a continuous relaxation to the Bernoulli variational posterior called the Binary Concrete Distribution [15]: $q_\Phi(\mathbf{A}) = \text{BinConcrete}(A_{ij}|\phi_{ij}, \tau)$, where τ is a temperature parameter that controls the strength of the relaxation. We also relax the prior in the same way in order to maintain a valid lower bound on the model evidence throughout optimization.

Once the optimal set of Θ and Φ parameters has been found, posterior inference of the labels can be carried out by sampling N_A graphs from $q_\Phi(\mathbf{A})$ and then computing the Monte Carlo sum given by equation 7.

Algorithm 2 VGCN

Input: $\mathbf{X}$, $\mathbf{Y}_{obs}$, $\mathbf{A}_{obs}$,
Output: p($\mathbf{Y}$ | $\mathbf{X}$,$\mathbf{Y}_{obs}$, $\mathbf{A}_{obs}$)

1: Compute prior $p(\mathbf{A})$ by smoothing $\mathbf{A}_{obs}$
2: Initialize Φ as the weights from $p(\mathbf{A})$
3: Find optimal parameters $\hat{\Phi}$ and $\hat{\Theta}$ via joint maximization of the ELBO using stochastic gradient ascent.
4: **for** i = 1 : $\mathbf{N}_\text{A}$ **do**
5: Sample graph $\mathbf{A}_i \sim q_{\hat{\Phi}}(\mathbf{A})$
6: Train GNN using graph $\mathbf{A}_i$
7: **for** j = 1 : $\mathbf{N}_\Theta$ **do**
8: Sample weights Θ_{ij} via MC dropout of $\hat{\Theta}$
9: Perform forward pass through the GNN using $\mathbf{A}_i$ and Θ_{ij}
10: Approximate p($\mathbf{Y}$ | $\mathbf{X}$,$\mathbf{Y}_{obs}$, $\mathbf{A}_{obs}$) by taking the average of all GNN outputs from step 7.

4.4 Comparison of Bayesian Models

The BGCN and VGCN take different approaches in modelling the data to produce their respective generative models over the graph. This will therefore lead to various differences in the properties of the graph samples from each model. In this section, we compare and contrast these differences and discuss the potential ramifications of each.

The generative model for the graph in the VGCN has a conditioning set that contains all the observed data ie $p(\mathbf{A} \mid \mathbf{X}, \mathbf{Y}_{obs}, \mathbf{A}_{obs})$. On the other hand, the BGCN's graph generative model is only conditioned on the observed graph ie $p(\mathbf{A} \mid \mathbf{A}_{obs})$. The node features and observed labels are not used here, instead the BGCN's approach is to find a MMSBM distribution that gives the maximum likelihood of realizing the original graph $\mathbf{A}_{obs}$ as a sample. The motivation behind graph structure learning is to uncover implicit

relational structure within the data itself and as the BGCN does not do this, it will learn a generative model that ignores this structure. This relational structure will therefore not necessarily be informative to the functional mapping on the data that one aims to learn.

The BGCN enforces a structure by ensuring that the generative model lies within the space of MMSBM distributions. Whereas the VGCN essentially enforces zero structure as each edge is independently distributed. The lack of constraints means that the search space of potential distributions is much larger for the VGCN. Specifically, the number of parameters in the MMSBM scales as $O(N)$ while the VGCN scales with $O(N^2)$. It has been widely demonstrated in machine learning that general models which leverage computing resources and data for training outperform more constrained models [17]. A major reason for this is that general models contain fewer prior assumptions on the nature of the data, many of which end up being incorrect. If however one is limited by computing resources or data, then too general models will be difficult or impossible to train, an issue known as the curse of dimensionality. If one has a fixed amount of data, then as the the number the number of parameters increases [1], the data will more and more sparsely cover the search space, making optimization increasingly difficult. We are attempting to train semi-supervised classification models on a single graph of up to ~ 30 thousand nodes, which is small compared to the size of the search space of ~ 500 million parameters with the VGCN. It therefore seems likely that the curse of dimensionality will lead to difficulties in the training procedure here.

Another difference between the models is the nature of the training procedure. The BGCN fully optimizes the generative model and then proceeds to train a ensemble GNN using samples from this distribution. On the other hand, the VGCN optimizes the generative model and the GNN in parallel. Refer to Figure 2 for a visualization of the stages. This parallel optimization allows feedback from changes to the graph generation process to propagate through the loss function and then be automatically adjusted as necessary. Separating the training procedure makes each section lower dimensional and easier to train however it leads to an intermediate representation (the graph generative model in this case) which is not necessarily optimal to the overall task.

In conclusion, the different approaches taken by the two Bayesian models have various benefits and drawbacks from a theoretical point of view. These differences will lead to variations in the properties of the graphs generated and as a result, differences in the performance of the classification functions learnt over them.

[1] Broadly, the generality of a model increases with the number of parameters it contains. This is because a greater flexibility in possible model instantiations is available the larger the parameter search space is.

5 Scalable Bayesian models

A common issue with Bayesian models is the significantly increased computational requirements compared to a non Bayesian equivalent. A major reason for this is that generative models introduce large numbers of hidden variables which must be inferred during training.

In general, graph generative models are dense distributions over the graph structure which causes them to have time and space complexities that scale with the number of possible edges, which is proportional to the square of the number of nodes. This leads to high dimensional parameter spaces which are difficult to optimize and store due to memory limitations on GPUs, limiting the models from scaling to large graphs. In their original papers, both the BGCN and VGCN were only tested on the benchmark graphs of about 3000 nodes. Most real world graphs are at least 10,000 nodes and therefore it is paramount that scalable models are developed in order for the real world application of Bayesian GSL. Our music genre graph is one such example, with 36876 nodes. In this section, we investigate some modifications to the VGCN model that give it a significant scalability boost via a reduction in the time and space complexity of both the training and inference stages.

5.1 Low Rank VGCN

As each edge in the free parameterization is independently parameterized, there are $\mathrm{N}(\mathrm{N}-1)/2$ latent variables in total. This prevents the model from scaling to larger graphs because the parameter set will become too large to be stored in a GPU memory during training. Additionally, the search space will become very high dimensional making it increasing difficult for optimization algorithms to find good directions to head in.

Another more compact representation that was discussed in [14] is a low rank parameterization for the variational posterior. The elementwise edge probabilities are given by:

$$\phi_{ij} = \sigma(\mathbf{z}_i^T \tilde{\mathbf{z}}_j + b_i + \tilde{b}_j + s), \quad \mathbf{z}_i, \tilde{\mathbf{z}}_j \in \mathbb{R}^{d_z}, \quad b_i, \tilde{b}_j, s \in \mathbb{R}, \quad \forall i, j = 1, ..., \mathrm{N}, \quad d_z < N \tag{13}$$

Our parameter set therefore consists of the matrix factorization $\tilde{\mathbf{Z}}$ and $\tilde{\mathbf{Z}}$, the bias vectors $\mathbf{b}$ and $\tilde{\mathbf{b}}$, and a scalar shift term s. $\sigma(a) = \frac{1}{1+e^{-a}}$ is a sigmoid function that ensures the edge probabilities remain in the range [0,1]. This leads to $N(2d_z + 1) + 1$ parameters which is a linear scaling in the number of nodes as opposed to the quadratic scaling that exists with the free parameterization. The product of the two

rank d_z matricies $\mathbf{Z}$ and $\tilde{\mathbf{Z}}$ is also a rank d_z matrix which means that our resulting adjacency matrix distribution will be rank d_z. The motivations for this model are from the belief that lower dimensional structure in the graph exists and therefore the majority of the variance can be explained by a small number of features.

An example to illustrate this fact would be community structure in graphs. A perfect community structure for $C << N$ communities in a graph would be if every node was fully connected with all other nodes in its community and none from any other community. If this was the case, then we would have C degrees of freedom and would therefore only need C variables to fully define the edge set for each node in the graph. Real world graphs will never be this perfect however they should exhibit some level of community structure and therefore a more compact parameterization should be give a reasonably accurate approximation of the true graph in many situations.

The initialization of the variational distribution is important to consider, as bad initializations may prevent a suitable direction to be found during the gradient based optimization. With the free parameterization, the obvious choice is to simply use the matrix of weights from smoothed prior $\mathbf{A}_s$. For the low rank case, we must initialize $\mathbf{Z}, \tilde{\mathbf{Z}}, \mathbf{b}, \tilde{\mathbf{b}}$, and s. We initialize $\mathbf{Z}, \tilde{\mathbf{Z}} \in \mathbb{R}^{N \times d_z}$ by finding a rank d_z approximation to the smoothed prior. This can be achieved via an approximate factorization of $\mathbf{A}_s \in \mathbb{R}^{N \times N}$ into two matricies. We then initialize $\mathbf{b}$, $\tilde{\mathbf{b}}$, and s to zero, which means the overall initial distribution given by (13) should approximate a freely parameterized distribution with the edge probabilities set to the smoothed prior $\mathbf{A}_s$.

Matrix factorization is in general not a one to one mapping and so a number of algorithms exist that optimize for different aspects in the resulting factorization [18]. The truncated singular value decomposition (T-SVD) gives the optimal rank d_z approximation to $\mathbf{A}_s$. Specifically, it produces an rank d_z approximation that minimizes the Frobenious norm of the residual between the true and approximate matricies. Note that if in fact the smoothed adjacency isn't a full rank matrix, but is instead rank $r < N$, then no information is lost using the low rank representation with $d_z \geq r$. The Randomized (T-SVD) [19] algorithm was used to compute the decomposition due to its ability to scale to large matrices.

$$\mathbf{A}_s \approx \mathbf{U}_{d_z} \Sigma_{d_z} \mathbf{V}_{d_z}^T = \mathbf{Z}\tilde{\mathbf{Z}}^T \qquad \text{where} \quad \mathbf{Z} = \mathbf{U}_{d_z} \Sigma_{d_z}, \quad \tilde{\mathbf{Z}} = \mathbf{V}_{d_z} \tag{14}$$

We computed the low rank approximations of the Spotify graph with the FMA medium subset, along with Cora and Citeseer, over the range $0 < d_z \leq N$. The Frobenious norm of the residual $\left| \mathbf{A}_s - \mathbf{Z}\tilde{\mathbf{Z}}^T \right|$

was then computed for the three graphs and plotted in Figure 7.

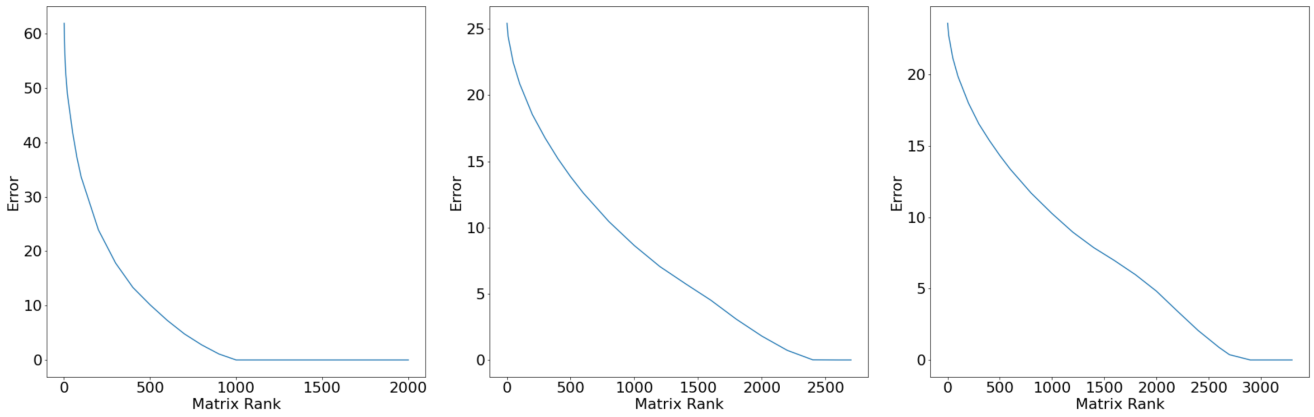

Figure 8: The frobenious norm of the residual (error) plotted against the rank of the approximation for the FMA medium subset (left), Cora (middle), Citeseer (right). Note that the FMA medium subset plot has been truncated.

For the FMA medium subset adjacency matrix, we observe a rapid drop off of the norm as the rank is increased, falling to zero at around 1000. The matrix has size 6288×6288, which shows that it is highly degenerate. Due to the rapid norm drop off, Roughly 80% of the variance in $\mathbf{A}_s$ can be described by 500 features. This analysis provides strong motivations for a low rank parameterization of the Spotify graph, with d_z in the range 500 to 1000. In contrast Cora and Citeseer exhibit a more gradual drop off of the norm, falling to zero at ranks close to the size of their respective adjacency matrices. This shows that these graphs are almost full rank and therefore low rank parameterizations are likely to incur significant information loss on them.

5.2 Top-K approximation

In order to make the distribution differentiable, the VGCN model substitutes the Bernoulli random variables for Binary Concrete ones, a continuous alternative. Weighted graphs are then sampled from this distribution rather than discrete ones if Bernoulli variables were used. Discrete graph adjacency matrices are in general sparse ($\epsilon << N^2$), where ϵ is the number of edges in the graph. This means that they can be stored efficiently using a sparse matrix representation with $O(\epsilon)$ space complexity. On the other hand, weighted graphs give rise to dense adjacency matrices. These have no way to be stored more compactly and therefore the space complexity of storing one is $O(N^2)$. GPU storage limitations prevent the weighted adjacency matrix from fitting in memory, which prevents the VGCN from scaling to large graphs of 10,000+ nodes.

Our solution is to sample a fully connected weighted graph and then take the top K edges, disregarding

the rest. A heap data structure is used to find the top K edges in the N × N array with an $O(N^2 \log K)$ runtime. One can use $K << N^2$, which leads to a graph with a sparse adjacency matrix that can fit in GPU memory. The motivation behind this approach is that the majority of the edges in the dense sample will have negligible weights, and therefore they can be approximated to zero with very little affect on the output of a forward pass of the GNN on the graph.

The size of the hyperparameter K sets the number of edges in the sampled graph. The optimal value of K for a specific graph will depend on the nature of the prior graph provided. In situations where there is noise in the prior graph, we desire a graph with fewer than ϵ edges since we hope that the generative model will learn to remove these uninformative edges. We therefore set $K < \epsilon$. On the other hand, if we expect there to be new relational structure learnt by the generative model, then we desire a graph with greater than epsilon edges. This can be achieved by setting $\epsilon < K << N^2$.

In this way, controlling K can be used to impose structural constraints on the graph generation process according to prior beliefs on the nature of the prior graph. If no prior belief exists in this regard, then one can simply set $K = \epsilon$. Alternatively, hyperparameter optimization can be used to find an optimal K.

5.3 Batched Implementation

The Top-K sparse approximation makes the GNN forward pass computationally viable, however one still has to store the full dense matrix in the intermediate stage before the top K are taken. Additionally, the dense adjacency is required to compute the KL divergence term in the ELBO. This again leads to an $O(N^2)$ space complexity so memory limitations of the GPU prevent this from being computationally viable. We propose a batched implementation of the VGCN that provides a significantly reduced space complexity.

Each edge in the VGCN variational distribution is independently distributed as a relaxed Bernoulli. This independence means that we can perform operations on parts of the distribution separately and then combine the results. The $N \times N$ distribution over the adjacency matrix can be separated into M^2 square chunks, where each chunk is $\frac{N}{M} \times \frac{N}{M}$. We denote this chunk of the adjacency matrix as $\mathbf{A}^{a,b} \ \ \forall a, b = 1, ..., M$, where a and b define the location of the chunk. The low rank model given in (13) can be computed separately for each chunk. We provide a visualization of this operation in Figure 8. We can then sample this distribution to obtain an adjacency matrix for the corresponding section of the graph. We approximate the number of the top K edges in the overall adjacency within each chunk as $\frac{K}{M^2}$. This leads to an $O(\frac{N^2}{M^2} \log \frac{k}{M^2})$ runtime to obtain the top edges, once found they are added to a

sparse matrix for storage. This approximation should be be accurate as long as the node ordering in the graph has been randomized, and a large enough value of M has been used. Finally, we compute the prior for the chunk and then use it to find the KL divergence between itself and the variational posterior for the chunk. All M^2 chunks are passed through and the resulting top edges of each chunk are stored in a sparse matrix. The overall KL divergence is simply the sum of the KL divergences for each chunk since the distributions are separable. The top K edges are then used to define a graph which gets passed into the GNN module to complete the forward pass through the model.

This chunk by chunk approach means that we never need to store an $N \times N$ dense matrix in memory, allowing the VGCN to scale to graphs that are orders of magnitude larger than what was possible with the standard model.

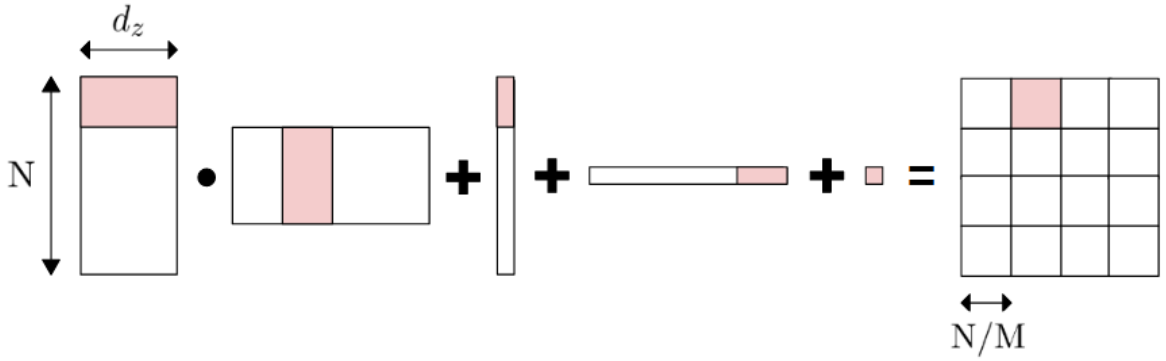

Figure 9: The low rank model for the variational posterior $\phi_{ij} = \mathbf{z}_i^T \tilde{\mathbf{z}}_j + b_i + b_j + s$ can be computed in a series of chunks. First the product of the relevant chunks of $\mathbf{Z}, \tilde{\mathbf{Z}}$ is found, then the segments of the bias vectors are added, and finally the shift term is added elementwise. We provide a visualization of how these chunks are produced in the figure. A sigmoid activation is then applied element wise to find the edge probabilities.

Algorithm 3 Low Rank Top K Batched VGCN - Training Procedure

Input: $\mathbf{X}$, $\mathbf{Y}_{obs}$, $\mathbf{A}_{obs}$,
Output: p($\mathbf{Y} \mid \mathbf{X}$,$\mathbf{Y}_{obs}$, $\mathbf{A}_{obs}$)

1: Compute prior $p(\mathbf{A})$ by smoothing $\mathbf{A}_{obs}$
2: Initialize Φ as the weights from $p(\mathbf{A})$
3: Initialize $\mathbf{Z}, \tilde{\mathbf{Z}}$ via R-TSVD on $\mathbf{A}_s$
4: **for** i = 1 : $\mathbf{N}_\mathrm{A}$ **do**
5: Initialize empty sparse adjacency matrix
6: **for** a = 1 : **M do**
7: **for** b = 1 : **M do**
8: Extract chunks of $\mathbf{Z}, \tilde{\mathbf{Z}}$
9: Compute low rank distribution for chunk $q_{\hat{\Phi}}(\mathbf{A}^{a,b})$
10: Sample a dense adjacency matrix $\mathbf{A}_i^{a,b} \sim q_{\hat{\Phi}}(\mathbf{A}^{a,b})$ from the distribution
11: Compute KL divergence between $q_{\hat{\Phi}}(\mathbf{A}^{a,b})$ and $p(\mathbf{A}^{a,b})$
12: Take top $\frac{K}{\mathbf{M}^2}$ edges from $\mathbf{A}_i^{a,b}$
13: Add these edges to the sparse adjacency matrix
14: **for** j = 1 : $\mathbf{N}_\Theta$ **do**
15: Sample weights Θ_{ij} via MC dropout of $\hat{\Theta}$
16: Perform forward pass through the GNN using $\mathbf{A}_i^{a,b}$ and Θ_{ij}
17: Approximate p($\mathbf{Y} \mid \mathbf{X}$,$\mathbf{Y}_{obs}$, $\mathbf{A}_{obs}$) by taking the average of all GNN outputs from step 7.

6 Experiments

6.1 Training Setup

In this section, we discuss the experiments taken with our algorithms along with the respective training details for each. Pytorch implementations of the algorithms were produced and used for the experiments, and a Tesla P100 PCIe 16 GB GPU was used for the training procedures. To make the results comparable, we used the same GCN module with all the Bayesin models. Motivated by the analysis in [6], we used a 2 layer architecture with a hidden layer of size 100. This hidden layer size seemed to max out the performance of the GCN, higher values led to increased training complexity with no significant improvements in classification accuracy. A 2 layer GAT was used with 8 attention heads and 8 features per head, giving 64 total hidden features which makes the architecture comparable to the GCN. We also trained a 2 layer MLP with 100 hidden features as a benchmark for comparison. During training, Dropout Nitish, et al. (2014) [27] was applied after the first layer with a dropout probability of 0.5 to reduce overfitting of the model.

Gradient based minimization was used to optimize all the objective functions present in the algorithms. The Adam optimizer Kingma (2014) [26] has been shown to be effective at finding robust minimas in high dimensional parameter spaces. We therefore use this algorithm for all of the optimizations carried out here. One only needs to set an initial learning rate, and then all other parameters are automatically set and adjusted as the iterations progress. L2 regularization of the GNN parameters with weight 0.0005 was used here to reduce overfitting of the network parameters.

For the freely parameterized VGCN, we used a similar hyperparameter setup to [14]. The smoothing factors α and β were set to 0.25 and 0.00005 respectively, which we felt suitably described our uncertainty in the original Spotify graph. To ensure that the relaxed Bernoulli prior accurately approximated the true Bernoulli distribution whilst still staying sufficiently continuous, we used a temperature of 0.1 (motivated by discussion in [15]). We attempted to set the temperature of the variational posterior to 0.1 too however this led to numerical instability in the KL divergence because initially the prior and posterior were identical. Instead, we used a temperature of 0.3. The model was very sensitive to the damping factor γ. Large values ($\gamma > 0.01$) made the KL divergence term in the ELBO dominate, preventing optimization of the likelihood term. Whilst small values ($\gamma < 0.0001$) made the model rapidly overfit to the training data as KL divergence term could effectively be ignored. The range $\gamma \in [0.001, 0.005]$ seemed to work as

	Graph Dataset			
	Cora	Citeseer	FMA medium subset	FMA large subset
Number of Nodes	2708	3327	6288	36876
Number of Edges	5429	4732	65626	1070592
Node Feature Size	1433	3703	140	140
Classification Type	Single-Label	Single-Label	Single-Label	Multi-Label
Number of Classes	7	6	16	161
Modularity	0.7963	0.877	0.961	0.950

Table 1: A summary of the Graph datasets used in this work.

a sufficient compromise for our data. A single Adam optimizer with an initial learning rate of 0.001 was used.

For the low rank batched VGCN, motivated by our analysis in section 4.1, we tested the algorithm with matrix ranks in the interval $d_z \in [0, 1000]$. For the top K approximation, we used a value of K equal to the number of nodes in the original graph. As each variational parameter affects many different values of the posterior distribution, the posterior is very sensitive to small changes in the parameters. For this reason, we use two separate Adam optimizers; one with an initial learning rate of 0.001 for the GNN parameters, and one with 0.0001 for the variational parameters. The same smoothing factors and temperatures as the free parameterization were used.

For the BGCN's MMSBM we used a cross community link probability δ of 0.0001. The Beta(ν) prior for the community strength of each community was set to $\nu = 1$ (perfect communities), and the Dirichlet(ζ) prior for the community membership distribution was set to $\zeta = \frac{1}{C}$ (uniform).

For the single label datasets, the problem can be framed as a multi class classification problem and therefore the negative log likelihood loss was used to train the GNN units. However, for the multi label case, this loss is no longer applicable. One is effectively attempting to train C binary classifiers on a single architecture, and therefore a more suitable loss is the binary cross entropy applied element wise between the predictions and the ground truth.

The model suited various different training procedures. The GCN and GAT required about 500 epochs for convergence. The BGCN required 10000 iterations for the optimization of the MMSBM, then 500 epochs to train each GCN in the Monte Carlo sum. Early stopping was used for the GCN here as there was a tendancy for the GCNs to overfit here. Both VGCN models required roughly 2000 epochs to converge.

	Graph Dataset			
	Cora	Citeseer	FMA medium subset	FMA large subset
MLP	55.1±0.3	46.5±0.4	71.2±0.2	0.68±0.003
GCN	81.4±0.5	70.9±0.5	85.5±0.4	0.82±0.004
GAT	81.6±0.7	71.0±0.7	86.2±0.4	0.83±0.002
BGCN	82.2±0.8	72.2±0.6	88.5±0.6	0.85±0.007
VGCN - Free	81.5±0.3	71.5±0.4	87.6±0.3	0.84±0.004
VGCN - Low Rank	81.3±0.9	70.2±1.1	87.2±0.7	0.83±0.005

Table 2: Classification performance of the various models. For the single label datasets (Cora, Citeseer, FMA medium subset), the metric is the percentage of correctly predicted labels. For the multi label case (FMA large subset), we use the ROC-AUC metric instead. Refer to section 5.2 for discussion on the choice of metrics.

6.2 Evaluation Metrics

For the single label case, we can evaluate the effectiveness of our algorithms by simply measuring the test accuracy, which is the percentage of correct predictions for songs in a test set. However for multi-label classification, the evaluation is not so straightforward. This is because one can have *partially* correct predictions, in addition to fully correct and incorrect ones. The Receiver Operating Characteristic (ROC) curve shows the performance of a binary classifier as its discrimination threshold is varied from 0 to 1 by plotting the true positive rate (TPR) against the false positive rate (FPR). The area under this curve (AUC) can be used as an evaluation metric for the binary classifier where a perfect classifier would have an AUC of 1 whilst a random classifier would have an AUC of 0.5. Multi-label classification is just a series of binary classifiers so the average of all the AUC for each label is sufficient metric for the overall performance of a classifier.

6.3 Results

Table 2 shows the results from our experiments. We observed large performance increases between the MLP and both the GCN and the GAT. This reproduces the results achieved by Sathyamurthy [4], validating the hypothesis that Geometric Deep Learning approaches are applicable to music genre classification and that the Spotify related artist structure is informative to the classification task. The GAT improves on the GCN by about 1%, which demonstrates the effectiveness of the attention mechanism here. All the Bayesian GSL models achieve performance increases over the basic GNNs. We see the strongest performance with the BGCN model, while the VGCN performs slightly worse. We provide some discussion on why this is the case in section 6.4.

Figure 9 shows the classification accuracy achieved on the medium subset with the Low Rank VGCN as a function of the rank d_z. We observe a rapid increase in accuracy with ranks in the range 0-500, after which the rate of increase decreases. The accuracy converged to a maximum of 87% by a rank of 1500, whilst we showed in section 5.1 that the rank of the graph adjacency was approximately 1000. This means that the variational posterior must have become higher dimensional than the prior, since otherwise, there would be no performance increase with the extra parameter dimensions. This suggests that the model was able to find new structure in the data that didn't exist in the prior.

The free parameterization of the VGCN was able to slightly outperform the low rank model. One potential cause is that the low rank model adds increased functional complexity through the low rank product, making it more difficult to optimize over.

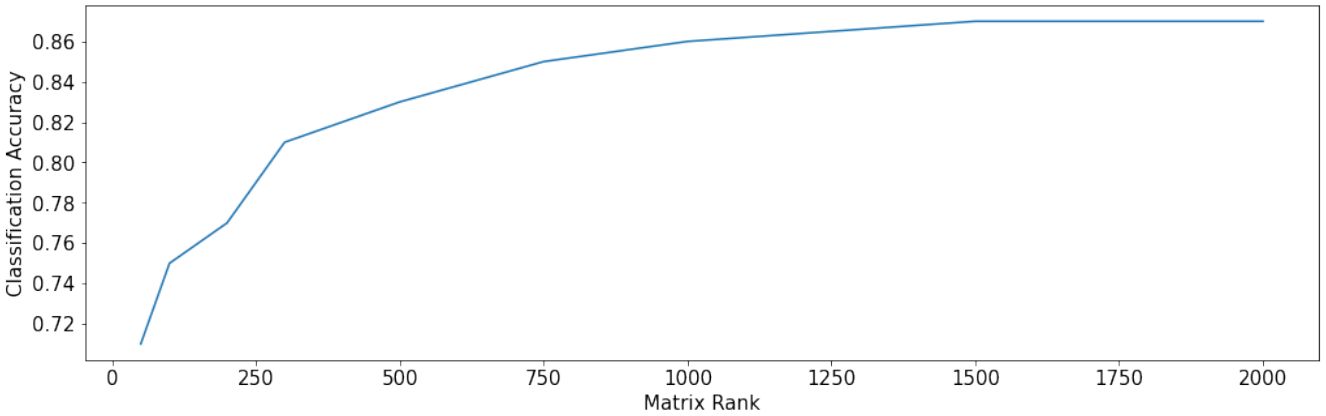

Figure 10: The relationship between the rank of the variational distribution over the graph adjacency matrix and the classification accuracy achieved using the low rank model on the FMA medium subset.

6.4 Graph Analysis

The different approaches taken by the algorithms will lead to different generative models for the graphs and therefore different realizations when these models are sampled. In this section, we compare the properties of the graphs produced by the models and attempt to motivate this in the theory of each. The FMA medium subset has 16 classes which we found led to it being more difficult to visualize the changes. For this reason we focused on the Citeseer dataset, this had just 6 labels allowing effective interpretation of changes to the graph structure.

In Figures 6.3 and 6.4, we display realizations from the trained generative models of BGCN and VGCN respectively. A force directed algorithm [22] was used to plot the graphs in a way that simultaneously minimised edge crossings and edge lengths. For comparison, the original Citeseer graph is shown in Figure 6.2. There are a number of visible changes to the structure. The BGCN was able to clearly

Figure 11: The original Citeseer graph.

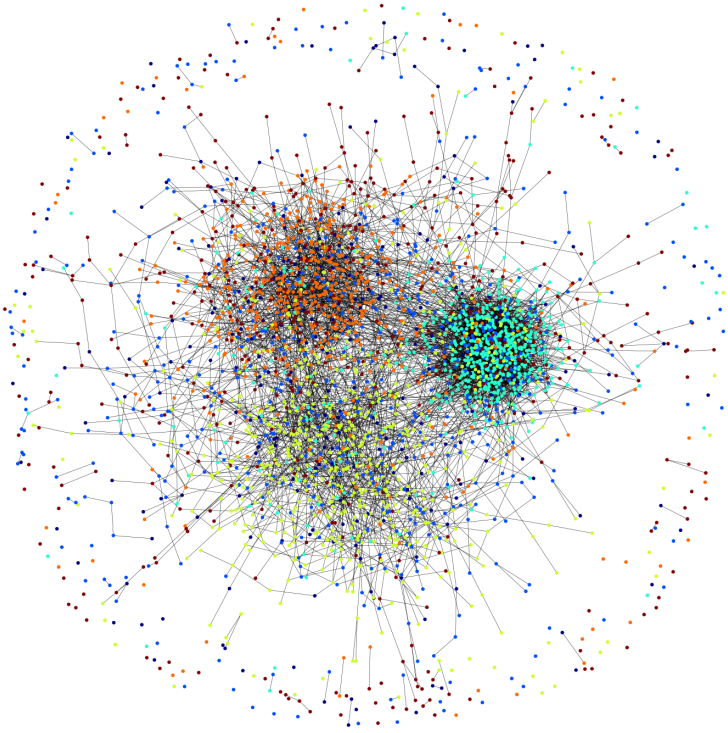

Figure 12: A realization from the graph generative model from the trained BGCN model with Citeseer.

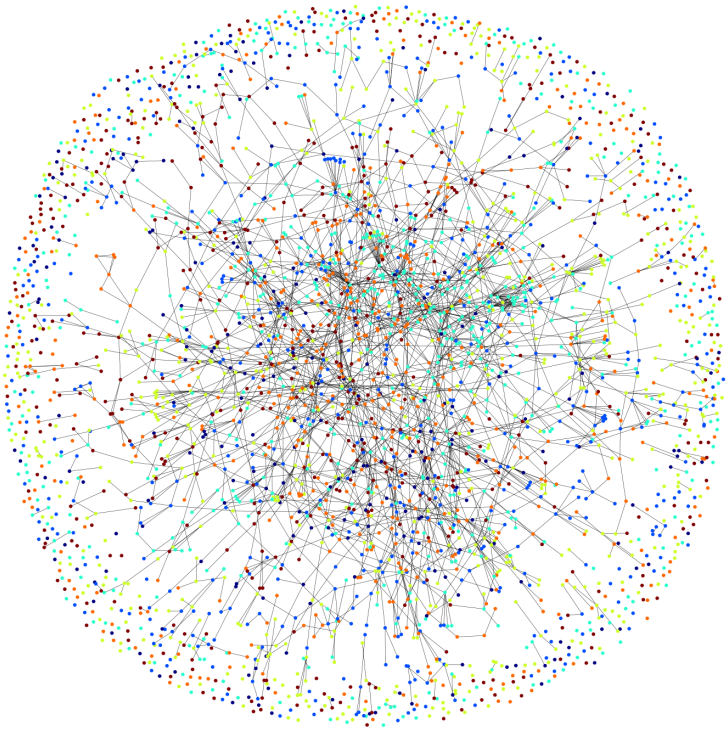

Figure 13: A realization from the graph generative model from the trained VGCN model with Citeseer.

improve the clustering of the classes in the graph. Most noticeably being the Cyan class, where almost perfect separation of the class from the rest could be achieved. The improvement in clustering of the classes was achieved despite the fact that the MMSBM model was not conditioned on any of the node features or observed labels. In section 4.4 we discussed the concern that the graph structure learnt by the MMSBM would not necessarily align with the labels, however these results suggest that the constraints imposed by the MMSBM were enough to learn a graph generative model that was informative for the classification task.

For the VGCN, a realization from the generative model gives a weighted graph rather than a discrete one. In order to visualize this we therefore find an approximate discrete graph by setting edges with a weight above 0.5 to 1 and the rest to 0. Structural changes are less visible here on a macro scale, but small clusters of about 5-10 nodes within the same class seem to be learnt quite regularly. This is likely to be caused by the free nature of the VGCN , where no structural constraints are imposed on the graph samples, making it more difficult for macro structure to emerge.

We also analyzed the degree distributions of the sampled graphs. We note that both the BGCN and VGCN shift the modal degree from 1 to 2. This reduces the number of stray nodes that are not in any sort of cluster at all, allowing one to utilize the graph structure to aid in their classification. As seen

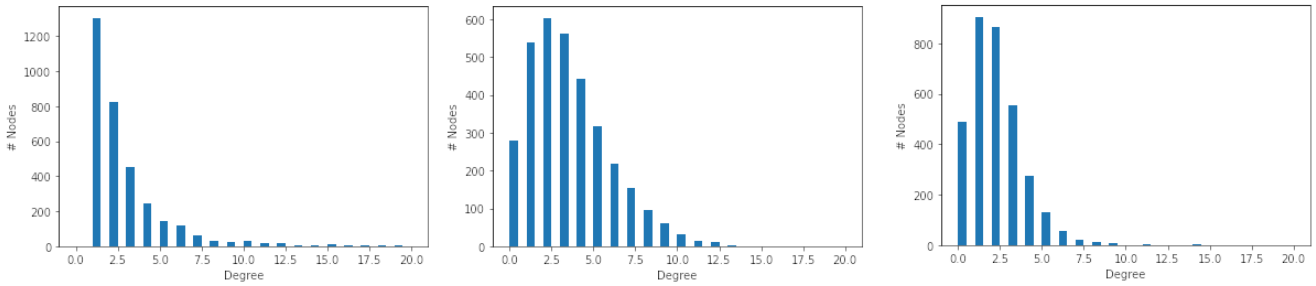

Figure 14: The degree distributions of samples from the Citeseer learnt graph distribution BGCN (middle), VGCN (right). The original Citeseer degree distribution is shown (left) for comparison.

in Figure 14, the original Citeseer degree distribution was long tailed, where a small number of nodes had a degree of up to 20. On the other hand, the learnt graph's degree distributions had shorter tails, decaying by a degree of about 12. GNN models have been shown [28] to overfit on graphs that have high degree nodes as the models can often end up relying too heavily on these nodes at the expense of the rest. This indicates that the learnt graphs have improved properties in this respect relative to the original. We also computed the average clustering coefficient for the graph samples, a measure which quantifies the number of triangles present as a metric. A value of 0.003 was found for the BGCN sample and 0.02 for the VGCN sample, compared to 0.14 for the original Citeseer graph. This decrease in clustering within the learnt graphs despite an increase in classification performance with them suggests that the extent of the clustering in the original graph was in some way detrimental to its usefulness in the task. One possible explanation for this is that the models were able to differentiate the highly informative clusters from the noisy ones. The noisy ones could be removed allowing the GNN model to use the clusters more effectively without overfitting to the noise.

In summary, we have shown how the different architectures for graph generative models leads to variations in the graph samples produced. The strong structural constraints imposed by the BGCN's MMSBM lead to visibly improved clustering of the classes relative to the original graph despite the MMSBM not being conditioned on any of the node features or observed labels. The VGCN was not able to perform so well which demonstrates the importance of imposing structure in the model architecture.

7 Conclusion

7.1 Summary of work

In this project we have presented Bayesian Graph Structure Learning approaches and applied them to the task of music genre classification. We focused on two core architectures; the BGCN and the VGCN. The algorithms were compared and contrasted, both in terms of their theoretical underpinnings as well as their experimental performance. Additionally, we highlighted scalability issues with the current methods, and then proposed a series of amendments to the VGCN model which attempt to mitigate these issues. The algorithms were able to outperform the baseline GNN architectures on the music genre task which demonstrates the effectiveness of graph structure learning for this problem. In particular, we found that the high degree of structure imposed by the MMSBM model in the BGCN allowed more optimal graph structures to be learnt in comparison to the VGCN's freely parameterized approach.

7.2 Future Directions

Bayesian Graph Structure Learning. As discussed in section 4.4, we see advantages and disadvantages to the approaches taken by both the BGCN and the VGCN. An ideal generative model for a graph would utilize strong structural constraints motivated by our prior beliefs (eg. the MMSBM for the BGCN), while fully utilizing all of the observed data (like the VGCN does). Additionally, we believe that it is more effective to optimize the generative model and the GNN parameters in parallel rather than in series so that some level of closed loop feedback can be achieved. Finally to increase scalability of the GSL models, sparse generative models that exploit the sparse nature of graph adjacency matrices should be investigated.

Music Genre Classification. This work treats the the label set as Euclidean, where the GNN embeds the graph into this Euclidean label space. However, as we discussed in section 2, the labels for music genres are in fact hierarchical and therefore they contain non-Euclidean structure within them. Our naive approaches here ignores this hierarchical nature and therefore misses out on important structure that could aid classification performance. Embedding the GNN output into hyperbolic space may be a more suitable representation for the labels which will allow one to more fully leverage the inherent structure in the data. The 2018 work from Wehrmann et al. [29], proposes a neural network architecture that allows hierarchical classification for multi label problems. Additionally, the 2019 work from Liu et

al. [30] proposes a Graph Neural Network for hyperbolic input spaces. We believe these two approaches could be used in future to build upon the current work applying Geometric Deep Learning to music genre classification.

References

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

[2] Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A. and Vandergheynst, P., 2017. Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine, 34(4), pp.18-42.

[3] Wilson, A.G., 2020. The case for Bayesian deep learning. arXiv preprint arXiv:2001.10995.

[4] Sathyamurthy, M., Dong, X. and Kumar, M.P., Geometric Deep Learning for Music Genre Classification. MML 2020, p.28.

[5] Spotipy.readthedocs.io. 2021. Welcome to Spotipy! — spotipy 2.0 documentation. [online] Available at: ¡https://spotipy.readthedocs.io/en/2.18.0/¿ [Accessed 9 May 2021].

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

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

[8] Hamilton, W.L., 2020. Graph representation learning. Synthesis Lectures on Artifical Intelligence and Machine Learning, 14(3), pp.1-159.

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

[10] Zhu, Y., Xu, W., Zhang, J., Liu, Q., Wu, S. and Wang, L., 2021. Deep Graph Structure Learning for Robust Representations: A Survey. arXiv preprint arXiv:2103.03036.

[11] Zheng, C., Zong, B., Cheng, W., Song, D., Ni, J., Yu, W., Chen, H. and Wang, W., 2020, November. Robust graph representation learning via neural sparsification. In International Conference on Machine Learning (pp. 11458-11468). PMLR.

[12] Li, R., Wang, S., Zhu, F. and Huang, J., 2018, April. Adaptive graph convolutional neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1).

[13] Gal, Y. and Ghahramani, Z., 2016, June. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning (pp. 1050-1059). PMLR.

[14] Elinas, P., Bonilla, E.V. and Tiao, L., 2019. Variational Inference for Graph Convolutional Networks in the Absence of Graph Data and Adversarial Settings. arXiv preprint arXiv:1906.01852.

[15] Maddison, C.J., Mnih, A. and Teh, Y.W., 2016. The concrete distribution: A continuous relaxation of discrete random variables. arXiv preprint arXiv:1611.00712.

[16] Zhang, Y., Pal, S., Coates, M. and Ustebay, D., 2019, July. Bayesian graph convolutional neural networks for semi-supervised classification. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, No. 01, pp. 5829-5836).

[17] Halevy, A., Norvig, P. and Pereira, F., 2009. The unreasonable effectiveness of data. IEEE Intelligent Systems, 24(2), pp.8-12.

[18] Koren, Y., Bell, R. and Volinsky, C., 2009. Matrix factorization techniques for recommender systems. Computer, 42(8), pp.30-37.

[19] Halko, N., Martinsson, P.G. and Tropp, J.A., 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM review, 53(2), pp.217-288.

[20] Oramas, S., Barbieri, F., Nieto Caballero, O. and Serra, X., 2018. Multimodal deep learning for music genre classification. Transactions of the International Society for Music Information Retrieval. 2018; 1 (1): 4-21.

[21] Defferrard, M., Benzi, K., Vandergheynst, P. and Bresson, X., 2016. Fma: A dataset for music analysis. arXiv preprint arXiv:1612.01840.

[22] Kobourov, S.G., 2012. Spring embedders and force directed graph drawing algorithms. arXiv preprint arXiv:1201.3011.

[23] McCallum, A., 2017. Cora dataset.

[24] Caragea, C., Wu, J., Ciobanu, A., Williams, K., Fernández-Ramírez, J., Chen, H.H., Wu, Z. and Giles, L., 2014, April. Citeseer x: A scholarly big dataset. In European Conference on Information Retrieval (pp. 311-322). Springer, Cham.

[25] Newman, M.E., 2006. Modularity and community structure in networks. Proceedings of the national academy of sciences, 103(23), pp.8577-8582.

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

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

[28] Garg, V., Jegelka, S. and Jaakkola, T., 2020, November. Generalization and representational limits of graph neural networks. In International Conference on Machine Learning (pp. 3419-3430). PMLR.

[29] Wehrmann, J., Cerri, R. and Barros, R., 2018, July. Hierarchical multi-label classification networks. In International Conference on Machine Learning (pp. 5075-5084). PMLR.

[30] Liu, Q., Nickel, M. and Kiela, D., 2019. Hyperbolic graph neural networks. arXiv preprint arXiv:1910.12892.