Flexible Variational Graph Auto-Encoders

Alex Kan[*]
Department of Computer Science
Department of Statistics and Operations Research
UNC-Chapel Hill
Chapel Hill, NC 25799
akan@unc.edu

Armaan Sethi
Department of Computer Science
Department of Physics and Astronomy
UNC-Chapel Hill
Chapel Hill, NC 27599
armaan@live.unc.edu

1 Introduction

In recent years, Graph Neural Networks (GNNs) have proved successful for the task of learning problems posed on non-grid structured data such as graphs, meshes, and point clouds. Recently, Kipf and Welling [12] have proposed a Graph Convolutional Network (GCN) layer that can be utilized for many learning tasks on graph represented data, including semi-supervised and unsupervised learning. These layers have the ability to learn flexible graph structure on the local level, and also can incorporate information gleaned from node features.

There has been much previous work on deep learning for graph-structured data. Graphs that represent objects and their relationships are ubiquitous in the real world. Social networks, e-commerce networks, biological networks may all be represented using graphs [21]. Graphs are usually complicated structures that contain rich underlying value.

Graph data can be extremely varied and contain diverse structures. There are many different properties of a graph that are important to consider, that are not represented in other data types. For example, graphs can be heterogeneous or homogeneous, weighted or unweighted, and signed or unsigned. Additionally, representation learning tasks on graphs can be either graph-focused, such as classification and generation, or node focused such as node classification and link prediction. These different tasks need different model architectures. Recently many new unsupervised methods such as Graph Autoencoders (GAEs), Graph Recurrent Neural Networks (Graph RNNs) and Graph Reinforcement learning have been used in order to learn the representation of graphs [21, 22].

Our work is currently focused on the link prediction problem with the realm of graph-structured data. We have chosen to expand on the Variational Autoencoder (VAE) [11] framework with the goal of developing a flexible model that can learn a good latent representation of undirected graphs and accurately predict their edges.

2 Related Work

2.1 Graph Convolutional Networks

There are several challenges with graph data that make using deep learning techniques nontrivial. While images, text, and audio all lie in a regular domain, graphs lie in an irregular domain making it hard to generalize many mathematical operations on graphs. One key example of this is that in a Convolutional Neural Network, it is not straightforward to define convolution and pooling operations on graphs.

In the paper "Semi-supervised Classification with Graph Convolutional Networks" (GCNs) [12] one mathematical definition of a graph convolutional operator was proposed and theoretically motivated:

[*]Project Github Repository may be found at: https://github.com/akan72/comp790

Table 1: Baseline Dataset Summary

Name	# of Nodes	# of Edges	Features per Node	# of Classes
Cora	2708	5429	1433	7
Citeseer	3327	4732	3703	6
Pubmed	19717	44338	500	3

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{\Theta},$$

Where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix with inserted self-loops and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ its diagonal degree matrix. $\mathbf{\Theta}$ is a layer-specific trainable weight matrix. After each layer an activation function, such as the ReLU is used.

Since the Graph Convolution is integral to our methods, it is necessary that we explain it in detail. Usually, when we talk about convolutions in Machine Learning we consider time series, 2D images, and occasionally 3D tensors. These can be thought of as special cases of learning on graphs, where the graph is a regular line, 2D square or 3D cube lattice with regular connectivity between neighboring pixels or tensors. Although this comparison works with regular graphs, it is not immediately obvious or intuitive how to generalize the concept of convolution to arbitrary graphs that may have nodes with very different local connectivity patterns or different degrees.

To generalize the concept, a Spectral view on convolutions is used. In the Fourier-domain, convolutions are simple pointwise multiplications of the Fourier-transform of a signal. Fourier-transforms generalize to graphs, therefore a general concept of graph convolutions as pointwise multiplication of the spectra of signals in the Fourier-domain can be defined. However, computing the spectrum of signals over general graphs involves matrix diagonalization, which generally has cubic complexity in the number of nodes. Computing exact convolutions over graphs is thus computationally extremely intensive, so approximations must be made by our Graph Convolution operation.

This convolution operation can be used in GCNs for semi-supervised learning by utilizing node attributes and node labels to train a model for a specific task such as classification. A first-order approximation of the spectral graph convolution allows models to learn hidden layer representations that encode both local graph structure and features of the nodes. There have also been many other similar graph convolution operators proposed such as the Chebyshev Spectral Graph Convolution operator from "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" [5], the graph neural network operator from "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks" [16], and the graph attentional operator from the "Graph Attention Networks"[18].

2.2 Other Graph Machine Learning Models

One improvement to graph convolutions is a graph attention network (GAT). As opposed to GCNs, GATs allows for implicitly assigning different importances to nodes of a same neighborhood, enabling a leap in model capacity. Analyzing the learned attentional weights may lead to benefits in interpretability, as was the case in the machine translation domain. This was achieved by leveraging masked self-attentional layers and stacking layers in order to provide attention to different neighborhoods' features.

In the paper "Variational Graph Auto-Encoders" [13] the authors create a Variational Graph Auto-Encoder (VGAE) based on a Variational Auto-Encoder (VAE) by using a Graph Convolutional Network (GCN) in the encoder and a simple inner product decoder for unsupervised learning on graph data. This method was able achieved competitive results on a link prediction task in citation networks.

Another way of generating graphs is by decomposing the graph generation process into a sequence of node and edge formations conditioned upon the graph structure generated so far. In the paper "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models" [20] the authors introduce a benchmark suite of datasets, baselines, and novel evaluation metrics to quantitatively evaluate the performance of graph generation.

3 Methods

3.1 Data

Our datasets are citation networks that come from PyTorch Geometric's "Datasets" package [7] which were introduced through the Planetoid framework [19]. We have ran our initial experiments on the 'Cora', 'Citeseer,' and 'Pubmed' datasets.

Each dataset contains an edgelist denoting citation links between documents. Within our implementation, the this edgelist is analogous to the previously defined adjacency matrix under the hood. These datasets of undirected, unweighted graphs, i.e. if the authors of document *i* cite document *j*, $A_{ij} = A_{ji} = 1$, regardless of the action of document *j*'s authors.

Feature vectors X, are comprised of bag-of-words representations of the documents that are cited. Although not currently utilized in our work, each dataset also includes a predicted class of document to which the node belongs. It is important to note that each of the previously stated three datasets consists of a single large graph. In our implementation, we are not utilizing any batching of our data, and are also simply holding out edges, rather than nodes.

During out initial experiments, we first applied our architecture to the MNIST Superpixels dataset [15] , but were unable to learn a flexible enough model to decrease our loss to a reasonable amount. Superpixel datasets represented in this fashion consist of graphs of non-overlapping groups of connected pixels of similar colors. Although the link prediction problem that we primarily focus upon does not have an implied ordering to our nodes, this may be important for other types of graph-represented data. For this type of data, the ordering of the nodes is of high importance. Without a particular ordering, the semantic meaning of a sample (such as the recognition of a handwritten MNIST digit) may be lost. We have chosen to not further explore this area at this time, because we feel that learning the "position" vectors necessary to the intrinsic meaning of MNIST digits is out of the scope of this project.

3.2 Task

Our main task within our project was performing link prediction. We chose this task because it was explored as a way of evaluating generated graphs in the original VGAE paper [13].

We utilize the Inference Network (in this case the Graph Convolutional Encoder) in order to learn a lower-dimensional representation of our graphs. The size of these embeddings corresponds to the number of nodes within the training graphs by the pre-selected number of codes that is specified by our architecture. For example, for the CORA dataset (a single graph with 2708 nodes) a a latent space of size 16, the embedding is of size 2708x16.

Our inference network as follows:

$$q(\mathbf{Z}|\mathbf{X},\mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z_i}|\mathbf{X},\mathbf{A}), \text{ where } (\mathbf{z_i}|\mathbf{X},\mathbf{A}) = \mathcal{N}(\mathbf{z_i}, \mu_\mathbf{i}, \text{diag}(\sigma_\mathbf{i}^\mathbf{2})$$

Our baseline generative model (with the inner product decoder) is as follows:

$$p(\mathbf{A}|\mathbf{Z} = \prod_{i=1}^{N}\prod_{j=1}^{N} p(A_{ij}|\mathbf{z_i},\mathbf{z_j}),$$
$$\text{where p(A}_{ij} = 1|\mathbf{z_i},\mathbf{z_j}) = \sigma(\mathbf{z_i}^T\mathbf{z_j})$$

These embeddings are then used to create a reconstructed adjacency matrix. In the simplest case, the reconstructed adjacency matrix results from a sigmoid function applied to the dot product of our embeddings and their transpose. This element A_ij can be intuitively interpreted as the probability of an undirected edge existing between node i and node j. These sigmoid probabilities are then used to map adjacency matrix elements to 1s and 0s for the final reconstruction.

3.3 Architecture

Our baseline model utilizes the same architecture as in Variational Graph Auto-Encoders [13]. The encoder has a single GCN layer which functions as a hidden layer, and then another GCN layer for each of the two variational parameters (μ and $\mathbf{\Sigma}$). For our baseline encoder network, we use a

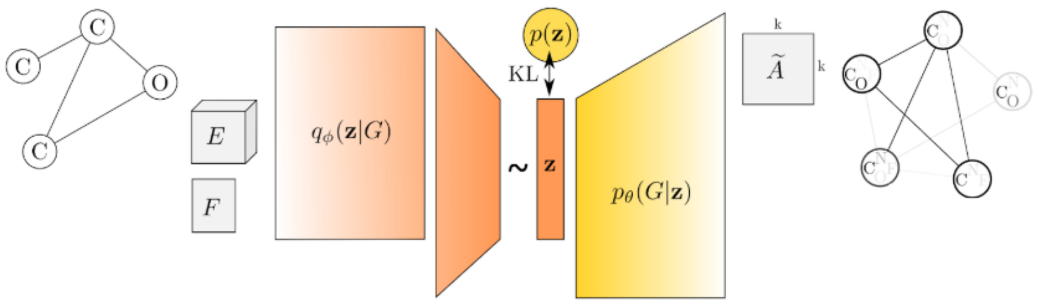

Figure 1: Illustration of a Variational Graph Autoencoder Model, based upon [17].

32-dimension hidden layer along with a 16-dimension latent code space per node. We explore adding more complex encoders by creating a deeper network with more GCN layers. Additionally, we explore the importance of the size of the latent space on the training of VGAE models by increasing the number of latent codes per node to 64-dimensions, and to 256-dimensions. For our baselines, our decoder network consists of a simple inner product between of our latent codes and their transpose [8], upon which a sigmoid function is applied. We also explore more complex decoders such as adding fully connected layers in our decoder and utilizing graph convolutions in our decoder.

We chose Adam as our optimizer [10] and used a learning rate of 0.01 as specified by the original VGAE paper [13]. We remove all diagonal entries from our original adjacency matrix and then randomly split all of the edges in our new matrix into train, test, and validation sets. Our training set contains 85% of the edges, validation has 5% of the edges, and test has 10%.We then normalize our adjacency matrix, and sparsify both it and our input features.

3.4 Loss

Because we can't directly minimize the KL divergence between $q_\phi(z)$ and $p_\theta(z|x)$, we can however, optimize an different objective function: the Evidence Lower Bound (ELBO). The minimization of the KL divergence between two distributions is the same as maximizing the ELBO [3]:

For our baseline model, we utilize Binary Cross Entropy (BCE) to quantify the reconstruction loss within our decoder network.

$$ELBO(q) = \mathbf{E}_q[\log p(x|z)] - \mathbf{E}_q[\log q_\lambda(z \mid x)]$$
$$\log p(x) = ELBO(q) + \mathbb{KL}(q(z) \mid\mid p(z \mid x))$$

During training, we take gradients with respect to our ELBO objective function and the normalize our training loss. We then evaluate the embeddings learned by our model on the held out sets of validation and test edges; we then compute the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve and the Average Precision (AP) of our predictions. These measures were chosen, as we are essential performing a binary classification on task on our adjacency matrix (whether or not an edge exists at each element).

3.5 Annealing

As stated in [9], "KL Cost Annealing" as originally formulated in [4] could be used in order to tackle the problem of "posterior collapse" within Variational Autoencoders. Posterior collapse occurs in a degenerate case when the posterior distribution learned by the model learns to imitate the prior distribution, ignoring the learned latent codes. This method consists of steadily increasing the weight of the KL divergence term within the ELBO from a usually small value (in our case taken to be 0) to the full weight of the KL divergence over a number of training epochs.

In our implementation, we linearly increase the weight over the KL regularizer from 0 to 1.0 over 10 training epochs.

Table 2: Baseline Dataset Results

Name	AUC	AP
Cora	91.4%	93.0%
Citeseer	89.3%	91.0%
Pubmed	94.3%	94.3%

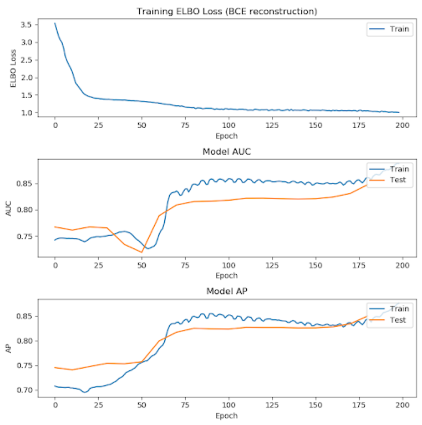

Figure 2: CITESEER Results with 1 Hidden GCN Layer.

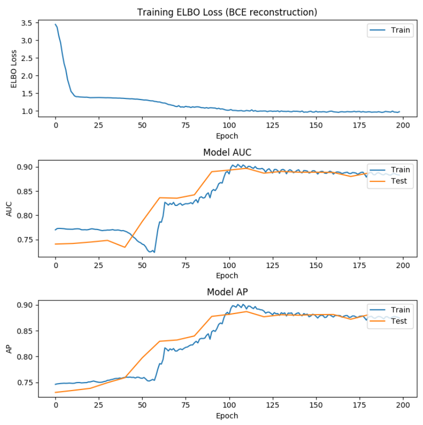

Figure 3: CITESEER Results with 2 Hidden GCN Layers.

3.6 Graph Similarity

As the KL divergence acts as a regularizer and is an intuitive measure of similarity between two probability distributions, another interesting avenue to explore would be to regularize our model with a Graph Similarity term. Several different measures of graph similarity exist including "Graph Edit Distance" which can be understood as an analogous to Edit Distance on strings. The Graph Edit Distance is defined as the minimum cost of elementary graph edit operations (node/edge, insertion, deletion and substituion) necessary to transform one graph into another.

The calculation of graph edit distance itself is very computationally intensive. However, there has been previous work on efficiently computing graph edit distance via approximation through Neural Networks. In [2], the authors propose SimGNN, a method that first maps graphs to a learned embedding vector (thought of as a high-level summary of the entire graph) and then performs node comparisons in a pairwise fashion to capture additional information at a about the low-level features the nodes within the graph. Similarity approximated through SimGNN is also preferable to that calculated by Maximum Common Subgraph (MCS) for similar reasons related to computational difficulty.

Another potential regularizer is the Graph Kernel [14]. Graph Kernels can as they are normally understood can be used to compute similarity between graphs, and can make use of commonly studied kernels like the Gaussian radial basis functional kernel.

However, these kernels must have fixed feature spaces, and are not differentiable (so they cannot be used in a loss function which allows for gradients to be taken).

As in Variational Graph Auto-Encoders paper, we trained our implementation for 200 epochs, using the Adam optimizer and a learning rate of 0.01. On the link prediction task, we believe that the current baseline model is able to learn a good representation of the undirected graphs in the Cora and Citeseer datasets. However, we do have concerns about our model's ability to correctly predict edges in the Pubmed dataset, as evidenced by the strange ROC curve produced.

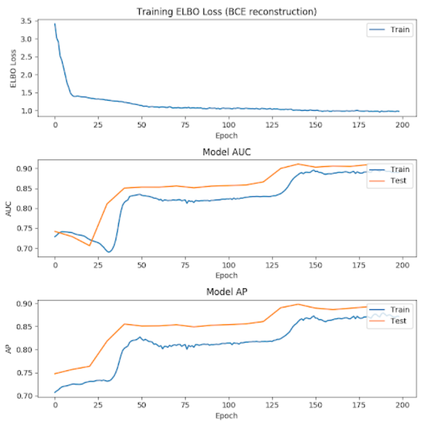

Figure 4: CITESEER Results with 3 Hidden GCN Layers.

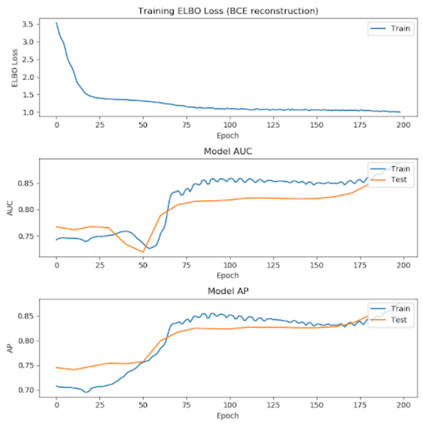

Figure 5: CITESEER Results with 16 code latent space.

4 Results

4.1 Architecture

By adding multiple GCN layers in the encoder, we tested the effects of deeper encoders in our model. We found that although the deeper networks performed better upon convergence, the benefit was minimal which can be seen in Figures 2,3, and 4. Contrarily, increasing the size of the latent space showed significant benefits. In Figures 5,6, and 7 we can see that by increasing the size of the latent space the model learned much faster. After about 100 epochs, there is a sizeable difference in the Model AUC and Model AP between the different models. Additionally, models with larger latent spaces converged to higher Model AUC and AP. Finally, we decided to explore a more flexible decoder capable of modeling complex dependencies present within our graph-structured data. We added a fully connected network to the decoder, but ran into many logistical issues. Since the size of our graphs were fairly large (on the order of thousands of nodes), and our model had a minimum of 16 latent codes, the fully connected layers in the decoder caused many issues with computational tractability. The final architectural change we tried was using upconvolutions in the decoder by utilizing GCN layers. We found this to work very poorly, which can be seen in Figure 9, but we believe it was due to the approximations made within the Graph Convolution operation.

One reason why graph convolutions in the decoder may not produce the expected results could be due to the approximations made within the Graph Convolution operation. The approximations used In order to make Graph Convolutions work involve limiting the set of convolution kernels under consideration. In [5] the authors express the convolutional kernel using Chebyshev polynomials of eigenvalues in the spectral domain to reduce the computational complexity of the convolution.

For example, if the convolutional kernel can be expressed or approximated using such Chebyshev polynomials, computing the convolution becomes easier. Like Taylor-expansion, this Chebyshev approximation can be made arbitrarily accurate by increasing the number of polynomials. Additionally, 1st order approximation in the Fourier domain restricts convolutions to kernels whose spectrum is an affine function of eigenvalues.

At all levels in this network, the filters are limited to three by three in size and are also essentially fixed to be the same kernel, up to a constant multiplier, across all layers and all units in the entire network.

Mathematically, our kernel can be described by the matrix:

$$\begin{vmatrix} \theta_1 c_1 & \theta_1 c_2 & \theta_1 c_1 \\ \theta_1 c_2 & \theta_0 + \theta_1 c_3 & \theta_1 c_2 \\ \theta_1 c_1 & \theta_1 c_2 & \theta_1 c_1 \end{vmatrix}$$

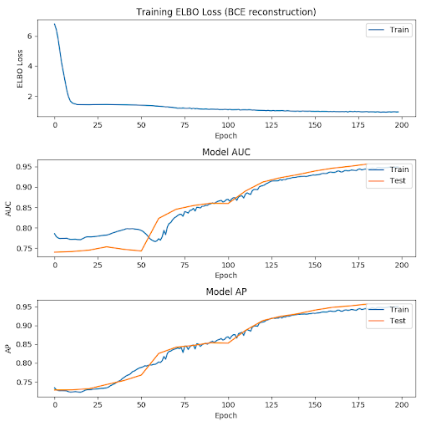

Figure 6: CITESEER Results with 64 code latent space.

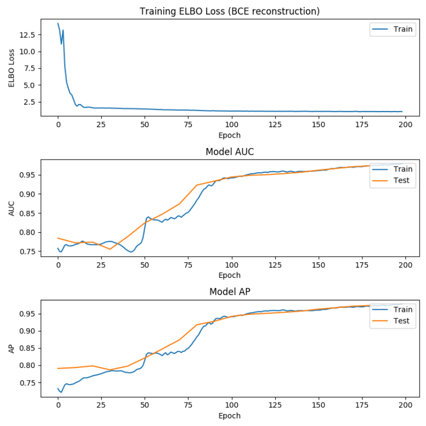

Figure 7: CITESEER Results with 256 code latent space.

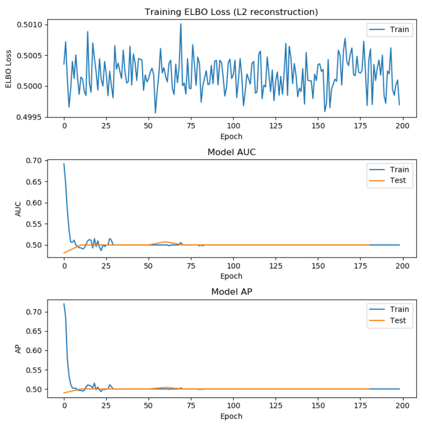

Figure 8: Failed L2 Loss.

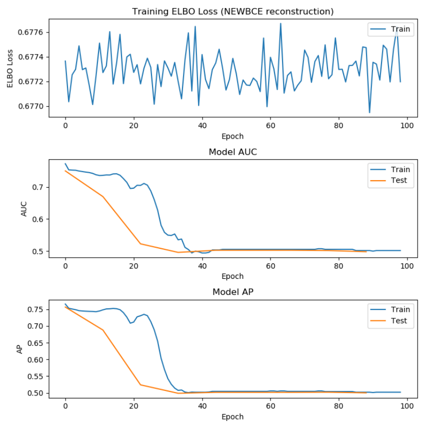

Figure 9: Results using GCN Layers in the Decoder network.

Within the kernel, c_1, c_2, and c_3 are fixed constants that depend on the graph weight parameters. The only trainable parameters are θ_0 and θ_1, and after all approximations $\theta_0 = -\theta_1$, so that the kernel only has a single scalar trainable parameter, essentially a scalar multiplier. Details on how c_1, c_2 and c_3 are computed aside, this model is in fact very, very limited in its flexibility.

4.2 Loss

We also tried to extend our VGAE model beyond using Binary Cross-Entropy to quantify reconstruction loss. A simple L2 reconstruction of our original adjacency matrix used towards this end. However, as in Figure 8 it can be shown that our model was unable to effectively learn using the L2 loss, and the predictions made by our classifer were also very poor. It is unclear whether this poor performance is due to a failure in implementation, of a lack of flexibility present in our model when an L2 loss is introduced over Binary Cross-Entropy.

4.3 Annealing

After applying our linear annealing schedule for a length of 10 epochs, we can see that for the CITSEER dataset, we have around a 5% improvement in performance in model AUC and AP. For CORA and PUBMED, we also see increases in performance for our two metrics, but with a lesser magnitude.

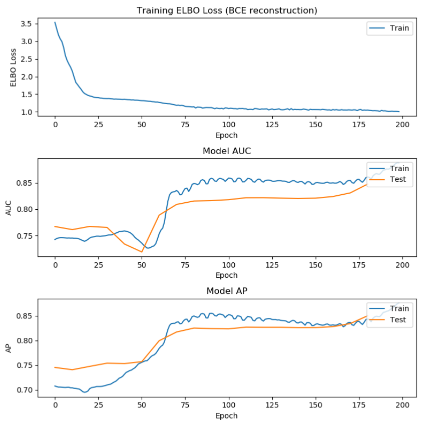

Figure 10: CITESEER Results without annealing.

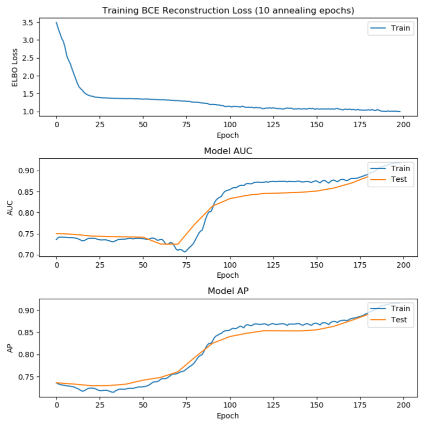

Figure 11: CITESEER Results with annealing.

5 Conclusion and Future Work

As we have seen, Variational Graph Auto-Encoder models using GCN layers are still early in their development, and they are not as robust or flexible as other VAE models applied to different modalities. Straightforward extensions of techniques commonly applied to domains such as images (upconvolutions, deeper encoder architectures, larger latent code sizes) may not necessarily improve performance as they do in images. Due to issues of computational complexity, these proposed extensions can be difficult to implement in practice. It may seem as though GCNs in graphs for link prediction may be as powerful as CNNs are in image processing, but the strict assumptions and inherent approximations associated with the use of GCNs may limit their flexibility.

For the link prediction task, good edge classification accuracy can be achieved and even improved using new methods such as KL Cost Annealing.

In order to counteract the computational complexity issues, batching could be introduced into our pipeline. The original VGAE implementation in [13] did not utilize batching, as the dataset required learning on the entire graph. However, learning on subgraphs of our large citation networks would allow us to test fully connected layers in the decoder due to the smaller latent code space. We believe that this would inject an additional level of robustness into our model, as we are currently only learning on held-out edges, but under this setting we could learn on heldout nodes, edges, or a combination thereof.

Similarly, we would like to apply our models to datasets comprising of multiple graphs rather than a single large graph. This would further reduce complexity, as adjacency matrices of much smaller size would have to be stored in memory, and the latent code spaces could be smaller. Additionally, our current problem formulation requires the presence of node features. It would be quite interesting to pursue similar tasks on graph-structured data lacking features, or to feed things other than bag-of-words feature vectors.

As highlighted in [13], "a Gaussian prior is potentially a poor choice in combination with an inner product decoder." This could prove interesting as the usage of different prior distributions could introduce a higher degree of flexibility into our model or potentially even alleviate the need to create a more complex decoder network. For example, a Student's t-distribution (with learned degrees of freedom) could be selected over a Gaussian prior if samples drawn from a heavier tailed distribution are desirable.

Although our L2 distance metric failed to produce meaningful results, we are interested to see how other losses such as Wasserstein/Earth Mover's Distance and Chamfer distance perform in this setting. Additional regularizers for our ELBO loss term could be interesting to pursue as well, especially in the case of a differentiable Graph Kernel variant.

If we desire additional experiments on new synthetic datasets, we plan to use longstanding graph generation algorithms (Erdos-Renyi [6], Barabasi-Albert [1]) to build additional benchmarking datasets. We believe that the Erdos-Renyi model would be another good benchmark because link generation is parameterized by a simple Bernoulli distribution. These datasets could also be an interesting extension, as data could be cheaply generated for the un-featurized setting.

An extremely promising model which could be used as an alternative to our inner-product decoder exists in the form of GraphRNN [20]. GraphRNN promises to model both graph-Level semantics at a high level and edge/node-level features at a low level. GraphRNN models are also a straightforward extension of our model because they may take in variable numbers of graphs as inputs during training. These models also work in the featureless setting and can be used for generating interpolations between different graph types.

References

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[2] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Graph edit distance computation via graph neural networks. *CoRR*, abs/1808.05689, 2018.

[3] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[4] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. *CoRR*, abs/1511.06349, 2015.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[6] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[7] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[8] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[9] Junxian He, Daniel Spokoyny, Graham Neubig, and Taylor Berg-Kirkpatrick. Lagging inference networks and posterior collapse in variational autoencoders. *CoRR*, abs/1901.05534, 2019.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[12] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

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

[14] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *CoRR*, abs/1903.11835, 2019.

[15] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *CoRR*, abs/1611.08402, 2016.

[16] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *arXiv preprint arXiv:1810.02244*, 2018.

[17] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. *CoRR*, abs/1802.03480, 2018.

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

[19] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[20] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: A deep generative model for graphs. *CoRR*, abs/1802.08773, 2018.

[21] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *CoRR*, abs/1812.04202, 2018.

[22] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.