Epitomic Variational Graph Autoencoder*

Rayyan Ahmad Khan\textsuperscript{1} and Martin Kleinsteuber\textsuperscript{2}

\textsuperscript{1} Technical University of Munich, Germany - rayyan.khan@tum.de
\textsuperscript{2} Mercateo AG, Munich, Germany - kleinsteuber@tum.de

Abstract. Variational autoencoder (VAE) is a widely used generative model for unsupervised learning of vector data. The learning capacity of VAE is often limited by over-pruning - a phenomenon that prevents many of the latent dimensions from learning any useful information about the input data. Variational graph autoencoder (VGAE) extends VAE for unsupervised learning of graph-structured data. Being an extension of VAE model, VGAE, also suffers from over-pruning in principal. In this paper we look at over-pruning in VGAE and observe that the generative capacity of VGAE is limited because of the way VGAE deals with this issue. We then propose epitomic variational graph autoencoder (EVGAE), a generative variational framework for graph datasets to overcome over-pruning. We show through experiments that the resulting model has a better generative ability and also achieves better scores in graph analysis related tasks.

Keywords: Graph autoencoder · Variational graph autoencoder · Graph neural networks · Over-pruning in graph neural networks · EVGAE.

1 Introduction

Graphs are data structures, used to model data points and the relations between them using nodes and edges respectively. A large number of real world problems can be expressed in terms of graphs e.g. protein-protein interactions\textsuperscript{6}, social and traffic networks\textsuperscript{9,11}, knowledge graphs\textsuperscript{8}, economic activity flows\textsuperscript{16} and recommender systems etc\textsuperscript{19,11,11}. Recently the field of deep learning has seen a noticeable growth in the interest in graph related problems because of increase in computational power, the power of graphs to model complex relations between objects and the ability of deep learning models to learn feature representations. Deep learning applications related to graphs include but are not limited to link prediction, node classification and clustering\textsuperscript{24,26}. These applications make use of the techniques that learn node representations while making use of the information in both node features and the structure of the graph. In the realm of unsupervised learning, variational graph autoencoder (VGAE)\textsuperscript{15} extends the variational autoencoder (VAE)\textsuperscript{5} model to encode the input data into a low dimensional space. Like VAE, VGAE tends to achieve the following two competitive objectives:

* Supported by Mercateo AG
An approximation of input data should be possible.

The latent representation of input data should follow standard gaussian distribution.

There is, however, a well-known issue with VAE in general: it is harsh in suppressing the embedding units that fail to capture enough information about the input data. Consequently, in practice, the number of units actually contributing to reconstruction of input data are less than the total available units in embedding layer. This issue is referred to as over-pruning [2], [21], [13]. As a result, we get an over-regularized model which limits its capacity. VGAE, being an extension of VAE for graph datasets, also suffers from over-pruning in embedding layer. For VAE, [2], [21], [13] and [25] introduce different approaches to tackle this issue. In this paper, we adopt a model based approach motivated by [25]. We consider our model to consist of multiple sparse VGAE models, called epitomes, that share the latent space such that for every graph node only one epitome is forced to follow standard gaussian distribution. This enables other latent dimensions to be more free in encoding useful information for the node. We first give a brief overview of VAE and VGAE and look into over-pruning in more detail. The section afterwards details epitomic variational graph autoencoder (EVGAE).

Towards the enc, we run some experiments on citation datasets [20] for proof of concept.

2 Variational Graph Autoencoder

Suppose that we are given an undirected and unweighted graph $G$ consisting of $N$ nodes $\{x_1, x_2, \cdots, x_N\}$ with each node having $F$ features. We assume that the node embeddings follow a certain predefined prior distribution, usually standard gaussian. Specifically, we assume that the information in nodes and edges can be jointly encoded in a $D$ dimensional latent feature space such that the respective random variables $\{z_1, z_2, \cdots, z_N\}$ follow standard gaussian distribution. These random variables are stacked into a matrix $Z \in \mathbb{R}^{N \times D}$. These latent representations are then fed to a decoder network to recover approximation of the input data. The assumption on latent representation allows the trained model to generate new data, similar to the training data, by sampling from the prior distribution. Following the same approach as VAE, we assume the joint distribution to be

$$p(G, Z) = p(Z)p(G|Z),$$

where

$$p(Z) = \prod_{i=0}^{N} p(z_i)$$

$$p(z_i) = \mathcal{N}(0, \text{diag}(1)) \forall i$$
We aim to learn the free parameters of our model such that our assumption is best met and we are able to recover graph information from the embeddings. For this, we choose to maximize log probability of $G$, i.e.

$$
\log(p(G)) = \log\left( \int p(Z)p(G|Z)\,dZ \right)
= \log\left( \int \frac{q(Z|G)}{q(Z|G)} p(Z)p(G|Z)\,dZ \right)
= \log\left( \mathbb{E}_{Z \sim q(Z|G)}\left\{ \frac{p(Z)p(G|Z)}{q(Z|G)} \right\} \right)
$$

(4)

Where $q(Z|G)$ is the approximate posterior to be learnt during model training. It is given by

$$
q(Z|G) = \prod_i q(z_i|G) \tag{5}
$$

$$
q(z_i|G) = \mathcal{N}\left( \mu_i(G), \text{diag}(\sigma_i^2(G)) \right) \tag{6}
$$

where $\mu_i(.)$ and $\sigma_i^2(.)$ are learnt using neural networks and samples of $q(Z|G)$ are obtained from mean and variance using reparameterization trick [5]. VGAE model uses graph convolution networks (GCN) [14] to learn $\mu_i(.)$ and $\sigma_i^2(.)$. GCN architecture focuses on the neighbors of the node under consideration rather than the whole graph $G$. To avoid overflows, we follow VAE and learn log variance instead of variance.

We make use of Jensen’s Inequality [23] to get ELBO bound of Eq. (4) and make it computationally tractable, i.e.

$$
\log(p(G)) \geq \mathbb{E}_{Z \sim q(Z|G)}\left\{ \log\left( \frac{p(Z)p(G|Z)}{q(Z|G)} \right) \right\} \tag{7}
= \mathbb{E}_{Z \sim q(Z|G)}\left\{ \log(p(G|Z)) \right\} + \mathbb{E}_{Z \sim q(Z|G)}\left\{ \log\left( \frac{p(Z)}{q(Z|G)} \right) \right\} \tag{8}
$$

The decoder network is given by $p(G|Z)$. For reconstruction, we limit ourselves to recover edge information from latent space. Since we are dealing with unweighted and undirected graphs, we can represent the edges information by an adjacency matrix $A \in \mathbb{R}^{N \times N}$ where $a_{ij}$ refers to the element in $i^{th}$ row and $j^{th}$ column. If an edge exists between node $i$ and $j$, we have $a_{ij} = 1$. The output $A$ is modeled by

$$
p(A|Z) = \prod_{(i,j)=(1,1)}^{(N,N)} p(a_{ij} = 1|z_i, z_j) \tag{9}
$$

$$
p(a_{ij} = 1|z_i, z_j) = \sigma(<z_i, z_j>) \tag{10}
$$
Where \(<, ., .>\) denotes dot product and \(\sigma(.)\) is the logistic sigmoid function. Hence the Eq. (8) can be re-written as

\[
\log(p(G)) = -BCE - D_{KL}(q(Z|G)||p(Z))
\]

(11)

Where \(BCE\) refers to binary cross-entropy between input edges and the reconstructed edges. The KL divergence is denoted by \(D_{KL}\). The respective loss, to be minimized, is

\[
L = BCE + D_{KL}(q(Z|G)||p(Z))
\]

(12)

3 Over-pruning in VGAE

VAEs in general suffer from over regularization caused by the second term of Eq.(12). This term forces the latent dimensions to follow standard gaussian distribution. So the dimensions which fail to encode enough information about input data are harshly penalized. In other words, if a latent dimension is adding little contribution to reconstruction, the loss function in Eq.(12) is better reduced by completely shutting that dimension down and simply generating standard gaussian noise along that dimension. As a result, these dimensions simply collapse to prior and fail to capture any information about the input data. Hence VAE fails to learn to its full capacity.

Since VGAE is built upon the same principle as VAE, it inherits the issue of over-pruning from VAE. We illustrate that with the help of an example. Let us use VGAE model to encode Cora dataset\(^3\). We use the same network architecture as given in the VGAE paper \(^15\) where mean and log-variance of 16 dimensional latent variable \(z\) are learnt using GCN networks. Using reparameterization trick\(^5\), we get the latent variables \(z\) in stacked form \(Z\) which is fed to inner product decoder and then to sigmoid to get \(A\). Fig. 1a shows the log variance of different components of latent space. We can see that except one component, all have converged to a value around 0.85. The plot of KL divergence along different dimensions is also drawn in a similar fashion. This plot also shows a similar behavior as shown in Fig. 1b where only one component seems to be responsible for encoding most of the information required for reconstruction i.e. only one component seems to be active. This implies that out of 16 components, the model is using only one component in true sense to encode input information. Thus VGAE in this case is utilizing only one sixteenth of its potential to encode input data. This notion was quantified in \(^3\). A latent unit is defined to be active if:

\[
A_u = \text{Cov}_x(\mathbb{E}_{u \sim q(u|z})\{u\}) \geq 10^{-2}
\]

\(^3\) Cora is a citation dataset\(^20\), widely used in graph analysis using deep learning. Details of the dataset given in experiments Sec. 5.1
Following this notion, we plot the unit activity along different dimensions of 16-unit VGAE in Fig. 1c. It is clear that only one unit is active i.e. far from prior. All others have become inactive and are not contributing to reconstruction. This problem is known as over-pruning. VAE has a tendency to over-penalize the latent units that fail to encode more information for reconstruction of the input. VGAE, being built on the same base model, inherits the same issue.

Reduction in contribution of KL divergence term is a simple way to get around this problem of over-pruning. i.e. we can add a weight term in the Eq. 8 as follows:

$$\log(p(G)) = \mathbb{E}_{Z \sim q(Z|G)} \left\{ \log(p(G|Z)) \right\} + \beta \mathbb{E}_{Z \sim q(Z|G)} \left\{ \log\left(\frac{p(Z)}{q(Z|G)}\right) \right\}$$  \text{(13)}

This approach can be linked with \(\beta\)VAE \[10,14\]. In \(\beta\)VAE, we choose \(\beta > 1\) to enforce better distribution matching. Conversely, if we select \(\beta < 1\), it puts less stress on distribution matching, allowing the latent distribution to be more different from standard gaussian. This allows the units more freedom for better reconstruction. For \(\beta = 0\), the model is reduced to simple non-variational autoencoder. This approach is adapted in VGAE paper \[15\] where \(\beta\) is chosen to be \((N)^{-1}\). Fig. 2 shows the evolution of KL divergence and unit activity with training epochs in this case. We can see that all the units are active now, participating in the reconstruction. Hence we get good reconstruction quality where all units are playing their part. However, the value of KL divergence is high along all the dimensions, signalling poor matching of \(q(Z|G)\) with standard gaussian. This directly affects the generative ability of model. We assumed the latent representation to follow standard gaussian distribution but the learnt distribution is not standard gaussian. So we cannot claim that random samples generated by standard gaussian and decoded by \(p(Z|G)\) will be similar to the graph data used for training. This approach to deal with over-pruning makes VAGE similar to its non-variational counter-part i.e. graph autoencoder (GAE).

Other existing solutions for VAE are also there. Adding dropout can also be another simple solution to achieve more active units. However this solution adds redundancy rather than encoding more useful information along different dimensions \[25\]. \[13\] proposes division of the latent units into subsets and forces each subset to contribute to the KL divergence. \[2\] uses KL cost annealing to activate more latent units. \[25\] uses a model based approach where latent units are divided into subsets with only one subset penalized for a certain data point. These subspaces also share some components which reduce redundancy between different subsets.

4 Epitomic Variational Graph Autoencoder

We now propose epitomic variational graph autoencoder (EVGAE) to deal with over-pruning in VGAE. The motivation comes from the observation that for a
Fig. 1: The values are smoothed out for better visuals. The actual values can still be seen in shadows. (a) and (b) show that only one out of 16 units is actively encoding input information required for reconstruction. This is confirmed by (c).
Fig. 2: The values are smoothed out for better visuals, as done in Fig. 1. The actual values can be seen in shadows. All the units are active but KL divergence is high, indicating poor matching of learnt distribution with prior, ultimately affecting generative ability of the model.

Fig. 3: Example of eight epitomes in a 16-dimensional latent space.
certain graph node, a subset of the latent space suffices to yield good reconstruction of edges. We assume $M$ subsets of the latent space called epitomes. They are denoted by $\{D_1, \cdots, D_M\}$. Furthermore, we ensure that every subset shares some components with at least one of other subsets. We intend to penalize only one epitome for an input node. This will encourage the other epitomes to be active. For this we define a discrete random variable $y_i$ that dictates which epitome is active for a given node $i$. The prior of $y_i$ is supposed to be uniform for all nodes. $y$ represents the stacked random vector for all $N$ nodes. So:

$$p(y) = \prod_i p(y_i); \quad p(y_i) = U(1, M)$$

(14)

Moreover we also define a binary matrix $E \in \mathbb{R}^{M \times D}$ which contains the information regarding which components belong to different subsets. Fig. 3 shows an example of $E$ with $M = 8$ in a 16-dimensional latent space where grayed components of $r^{th}$ row show the components that constitute the subset $D_r$. We denote $r^{th}$ row of $E$ by $E[r, :]$.

**Generative Model** of EVGAE is the same as that of VGAE i.e.:

$$p(G, Z) = p(Z) p(G | Z)$$

(15)

**Inference Model** uses the following approximate posterior:

$$q(Z, y|G) = q(y|G) q(Z|G)$$

(16)

$$q(y|G) = \prod_{i=1}^N q(y_i|G); \quad q(y_i|G) = \text{Cat}(\pi_i(G))$$

(17)

$$q(Z|G) = \prod_{i} q(z_i|G); \quad q(z_i|G) = N(\mu_i(G), \text{diag}(\sigma_i^2(G)))$$

(18)

Where $\text{Cat}(\cdot)$ refers to categorical distribution. $\pi_i(\cdot)$, $\mu_i(\cdot)$ and $\sigma_i^2(\cdot)$ are learnt using neural networks. Under the assumption that given $Z$, $y$ and $G$ are independent, the objective function to be maximized becomes

$$\log(p(G)) = \log\left(\int \sum_y p(y)p(Z|y)p(G|Z) \, dZ\right)$$

(19)

$$= \log\left(E_{(Z,y)\sim q(Z,y|G)}\left\{\frac{p(y)p(Z|y)p(G|Z)}{q(Z,y|G)}\right\}\right)$$

(20)

$$= \log\left(E_{(Z,y)\sim q(Z,y|G)}\left\{\frac{p(y)p(Z|y)p(G|Z)}{q(Z|G)q(y|G)}\right\}\right)$$

(21)
By using Jensen’s inequality \[23\], the ELBO bound for log probability becomes

\[
\log(p(G)) \geq \mathbb{E}_{(Z,y) \sim q(Z,y|G)} \left\{ \log \left( \frac{p(y)p(Z|y)p(G|Z)}{q(Z|G)q(y|G)} \right) \right\} \tag{22}
\]

\[
= \mathbb{E}_{Z \sim q(Z|G)} \left\{ \log \left( \frac{p(G|Z)}{q(G|Z)} \right) \right\} \\
+ \mathbb{E}_{y \sim q(y|G)} \left\{ \log \left( \frac{p(y)}{q(y|G)} \right) \right\} \\
+ \mathbb{E}_{(Z,y) \sim q(Z,y|G)} \left\{ \log \left( \frac{p(Z|y)}{q(Z|G)} \right) \right\} \tag{23}
\]

Like in the case of VGAE, we limit ourselves to recover edge information from latent space where the edges are unweighted and undirected. So the decoder is the same as in Eq. (10). Therefore the first term is the same as in VGAE i.e. binary cross-entropy between input and reconstructed edges. For this term, in addition to the positive samples given by \(A\), we also sample the negative edges, considering the remaining pairs of \(A\) as negative pairs. The number of negative edges is kept the same as that of positive edges to balance the training examples.

The second term is computed as:

\[
\mathbb{E}_{y \sim q(y|G)} \left\{ \log \left( \frac{p(y)}{q(y|G)} \right) \right\} = \mathbb{E}_{y \sim q(y|G)} \left\{ \sum_{i=1}^{N} \log \left( \frac{p(y_i)}{q(y_i|G)} \right) \right\} \\
= \sum_{i=1}^{N} \mathbb{E}_{y_i \sim q(y_i|G)} \left\{ \log \left( \frac{p(y_i)}{q(y_i|G)} \right) \right\} \\
= -\sum_{i=1}^{N} D_{KL} \left( q(y_i|G) \mid \mid p(y_i) \right) \\
= -\sum_{i=1}^{N} D_{KL} \left( \text{Cat}(\pi_i(G)) \mid \mid \mathcal{U}(1,M) \right) \tag{24}
\]

The third term is computed as follows:

\[
\mathbb{E}_{(Z,y) \sim q(Z,y|G)} \left\{ \log \left( \frac{p(Z|y)}{q(Z|G)} \right) \right\} \\
= \mathbb{E}_{y \sim q(y|G)} \left\{ \mathbb{E}_{Z \sim q(Z|G)} \left\{ \log \left( \frac{p(Z|y)}{q(Z|G)} \right) \right\} \right\} \\
= \sum_{y} q(y|G) \mathbb{E}_{Z \sim q(Z|G)} \left\{ \log \left( \frac{p(Z|y)}{q(Z|G)} \right) \right\} \\
= \sum_{y} q(y|G) \mathbb{E}_{z_i \sim q(z_i|G)} \left\{ \log \left( \frac{p(z_i|y_i)}{q(z_i|G)} \right) \right\} \\
= \sum_{y_i} q(y_i|G) \mathbb{E}_{z_i \sim q(z_i|G)} \left\{ \log \left( \frac{p(z_i|y_i)}{q(z_i|G)} \right) \right\} \tag{25}
\]
We take motivation from [25] to compute (25) as:

\[
\sum_{i=1}^{N} \sum_{y_i} q(y_i|G) \mathbb{E}_{z_i \sim q(z_i|G)} \left\{ \log \left( \frac{p(z_i|y_i)}{q(z_i|G)} \right) \right\}
\]

\[
= \sum_{i=1}^{N} \sum_{y_i} q(y_i|G) E[y_i, :] \odot \mathbb{E}_{z_i \sim q(z_i|G)} \left\{ \log \left( \frac{p(z_i)}{q(z_i|G)} \right) \right\}
\]

\[
= - \sum_{i=1}^{N} \sum_{y_i} q(y_i|G) E[y_i, :] \odot D_{KL} \left( q(z_i|G) || p(z) \right)
\]

\[
= - \sum_{i=1}^{N} \sum_{y_i} \pi_i(G) E[y_i, :] \odot D_{KL} \left( N(\mu_i(G), \text{diag}(\sigma_i^2(G))) || N(0, \text{diag}(1)) \right) \quad (26)
\]

i.e for every node, we sum over all epitomes. For a given epitome, we only consider the effect of dimensions activated in case of that epitome. The term ignores the dimensions that are not dictated by the given epitome. The remaining dimensions have freedom to better learn the reconstruction. As a result, EVGAE encourages more units to be active without penalizing the low performing units prematurely. It can be observed that for a single epitome with \( E[0, :] = 1 \), VGAE model is recovered. Hence the model generalizes VGAE. The algorithm for training EVGAE is given in [4] The final loss function, to be minimized, is given by:

\[
L = BCE + \sum_{i=1}^{N} D_{KL} \left( \text{Cat}(\pi_i(G)) || \mathcal{U}(1, M) \right)
\]

\[
+ \sum_{i=1}^{N} \sum_{y_i} \pi_i(G) E[y_i, :] \odot D_{KL} \left( N(\mu_i(G), \text{diag}(\sigma_i^2(G))) || N(0, \text{diag}(1)) \right) \quad (27)
\]

The code for EVGAE is available on github [4]

| Algorithm 1: EVGAE Algorithm |
|-------------------------------|
| **Input:**                  |
| - \( G \)                  |
| - Epochs                   |
| - The matrix \( E \) to dictate active dimensions for each epitome. |
|                             |
| Initialize model weights; \( i = 1 \) |
| **while** \( e \leq \text{Epochs} \) **do** |
|     compute \( \pi_i(\cdot), \mu_i(\cdot) \) and \( \sigma_i^2(\cdot) \) \( \forall i \); |
|     compute \( \mathbf{z}_i \) \( \forall i \) by reparameterization trick; |
|     compute loss using Eq. (27); |
|     update model weights using back propagation |
| **end** |

[4] https://github.com/RayyanRiaz/EVGAE/tree/master
5 Experiments

We start by comparing the example of Cora dataset, encoded using EVGAE, with Fig. 1 and Fig. 2. The number of latent dimensions is kept 16 as in VGAE. In addition, we select 8 epitomes, each dictating three units to be active with neighboring epitomes sharing one unit. This is the same assembly as shown in the Fig. 3. Fig. 4 shows the evolution of KL divergence and unit activity with training epochs. By comparing the figures, we can observe that EVGAE results in more units being active compared to VGAE. When stress on KL divergence is reduced as done in [15], all the units become active as shown in Fig. 2. However, this results in the latent distribution being less matched with the standard gaussian prior. This can be observed by comparing Fig. 2a with Fig. 4a. Hence we can state that although EVGAE results in less units being active, the distribution matching is better compared to VGAE with less stress on KL divergence.

![KL divergence along different latent dimensions in EVGAE.](image1)

(a) KL divergence along different latent dimensions in EVGAE.

![Unit activity along different latent dimensions of EVGAE.](image2)

(b) Unit activity along different latent dimensions of EVGAE.

Fig. 4: kl divergence and unit activity of EVGAE for Cora dataset
5.1 Datasets

We target the following citation datasets\cite{20} to compare the performance of EVGAE with VGAE on the link prediction:

**Cora** dataset has 2,708 nodes with 5,297 undirected and unweighted links. The nodes are defined by 1433 dimensional binary feature vectors, divided in 7 classes.

**Citeseer** dataset has 3,312 nodes defined by 3703 dimensional feature vectors. The nodes are divided in 6 distinct classes. There are 4,732 links between the nodes.

**PubMed** consists of 19,717 nodes defined by 500 dimensional feature vectors linked by 44,338 unweighted and undirected edges. These nodes are divided in 3 classes.

5.2 Implementation Details

We train the model on an incomplete version of each dataset i.e. the edges of each dataset are divided in training set, validation set and test set. We use 85% links for training, 5% for validation and 10% for testing the performance of the model. The validation set is used to optimize hyper-parameters of the model. During training, we use the training set to get positive edges. An equal number of negative/non-connected edges is obtained by sampling the remaining pairs of nodes. Testing phase classifies the test set as positive or negative edges. We compare the results with both simple and variational versions of graph auto-encoder\cite{15}. For VGAE, we also add the case with no stress on kl-divergence term. In addition to that, we compare with spectral clustering\cite{22} and DeepWalk\cite{18} as they all generate latent embeddings of graph nodes. However, VGAE, GAE and EVGAE have an advantage over DeepWalk and spectral clustering in a sense that we can also make use of the node features.

The implementation of spectral clustering is taken from \cite{17} with 128 dimensional embedding. For DeepWalk, the standard implementation is used as described in \cite{7}. For VGAE and GAE, we use the same implementation details as given by the authors in \cite{15}. EVGAE also follows a similar structure with latent embedding being 512 dimensional and the hidden layer consisting of 1,024 units, half of which give us $\mu_i(.)$ and the other half give log-variance. We select 256 epitomes for all three datasets. Each epitome enforces three units to be active, while sharing one unit with neighboring epitomes. This can also be viewed as an extension of the matrix shown in Fig. 3. Adam\cite{12} is used as optimizer with learning rate $1e^{-3}$. We can observe from Table I that the results of EVGAE are competitive or better than those of GAE/VGAE.

We now look at the number of active units for different latent dimensions. Fig. 5a gives an overview of active units for Cora dataset. We plot the active units for $D \in \{16, 32, 64, 128, 256, 512\}$. For all values of $D$, the number of epitomes is set to $\frac{D}{4}$ and one unit is set to overlap with each of neighboring epitomes just as done in case of $D = 16$ in Fig. 3. Note that this is just a choice we have made consistently in our work. EVGAE allows us to choose epitomes at will as long
Table 1: Results of link prediction on citation datasets

| Method          | Cora | Citeseer | PubMed |
|-----------------|------|----------|--------|
|                 | AUC  | AP       | AUC    | AP   | AUC  | AP   |
| DeepWalk        | 83.1 ± 0.01 | 85.0 ± 0.00 | 80.5 ± 0.02 | 83.6 ± 0.01 | 84.4 ± 0.00 | 84.1 ± 0.0 |
| Spectral Clustering | 84.6 ± 0.01 | 88.5 ± 0.00 | 80.5 ± 0.01 | 85.0 ± 0.01 | 84.2 ± 0.02 | 87.8 ± 0.01 |
| GAE             | 91.0 ± 0.02 | 92.0 ± 0.03 | 89.5 ± 0.04 | 89.9 ± 0.05 | 96.4 ± 0.00 | 96.5 ± 0.0 |
| VGAE w/o β     | 79.44 ± 0.03 | 80.51 ± 0.02 | 77.08 ± 0.03 | 79.07 ± 0.02 | 82.79 ± 0.01 | 83.88 ± 0.01 |
| EVGAE           | 92.96 ± 0.02 | 93.58 ± 0.03 | 91.55 ± 0.03 | 93.24 ± 0.02 | 96.80 ± 0.01 | 96.91 ± 0.02 |

as the basic rule of sharing the subspace is followed. It can be observed that with $\beta = 1$, the number of active units is quite less compared to the available units. This is because of the over penalizing nature of VAE in general. When we select $\beta = N^{-1}$ as done in [15], all the units get active irrespective of the latent dimension. In case of EVGAE, the number of active units is in between the two, i.e. we are able to mitigate the over-pruning while keeping $\beta = 1$. This results in better performance in graph analysis tasks as shown in table 1 while having a better distribution matching. To show distribution matching, we compare the average KL divergence of active units for different latent space dimensions. Only active units are considered when averaging the KL divergence otherwise the inactive units bias the results towards zero. Fig. 5b shows how the activations change as we increase the number of latent dimensions. We can see that when $\beta = 1$, the average KL divergence for active units is still quite small, resulting in a good matching between inferred distribution and the prior. Conversely, when $\beta = N^{-1}$ the average KL divergence per active component is quite high, pointing out that the learnt latent representation has a quite different learnt distribution compared to the prior. In case of EVGAE, the KL divergence is quite closer to the prior compared to VGAE. For $D = 512$, it is almost similar to the case with $\beta = 1$.

6 Conclusion

In this paper we have looked at the issue of over-pruning in variational graph autoencoder. We have shown that the way VGAE [15] deals with this issue results in a latent distribution that is quite different from the standard gaussian prior. We have proposed an alternative model based approach that mitigates the problem of over-pruning, encouraging more latent dimensions to actively play their role in reconstruction, while achieving a better matching between inferred and prior distribution of latent variable and also achieving competitive/better results compared to VGAE on link prediction tasks. The code for EVGAE is available on github.

5 https://github.com/RayyanRiaz/EVGAE
(a) Active units for different latent dimensions

(b) Average KL divergence of active units for different latent dimensions
References

1. Battaglia, P., Pascanu, R., Lai, M., Rezende, D.J., et al.: Interaction networks for learning about objects, relations and physics. In: Advances in neural information processing systems. pp. 4502–4510 (2016)

2. Bowman, S.R., Vilnis, L., Vinyals, O., Dai, A.M., Jozefowicz, R., Bengio, S.: Generating sentences from a continuous space. arXiv preprint arXiv:1511.06349 (2015)

3. Burda, Y., Grosse, R., Salakhutdinov, R.: Importance weighted autoencoders. arXiv preprint arXiv:1509.00519 (2015)

4. Burgess, C.P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., Lerchner, A.: Understanding disentangling in beta-vae. arXiv preprint arXiv:1804.03599 (2018)

5. Doersch, C.: Tutorial on variational autoencoders. arXiv preprint arXiv:1606.05908 (2016)

6. Fout, A., Byrd, J., Shariat, B., Ben-Hur, A.: Protein interface prediction using graph convolutional networks. In: Advances in neural information processing systems. pp. 6530–6539 (2017)

7. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864 (2016)

8. Hamaguchi, T., Oiwa, H., Shimbo, M., Matsumoto, Y.: Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. arXiv preprint arXiv:1706.05674 (2017)

9. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in neural information processing systems. pp. 1024–1034 (2017)

10. Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., Lerchner, A.: beta-vae: Learning basic visual concepts with a constrained variational framework. Iclr 2(5), 6 (2017)

11. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems. pp. 6348–6358 (2017)

12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

13. Kingma, D.P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., Welling, M.: Improved variational inference with inverse autoregressive flow. In: Advances in neural information processing systems. pp. 4743–4751 (2016)

14. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)

15. Kipf, T.N., Welling, M.: Variational graph auto-encoders. arXiv preprint arXiv:1611.07308 (2016)

16. Marques, A.G., Segarra, S., Leus, G., Ribeiro, A.: Sampling of graph signals with successive local aggregations. IEEE Transactions on Signal Processing 64(7), 1832–1843 (2015)

17. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. Journal of machine learning research 12(Oct), 2825–2830 (2011)

18. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710 (2014)
19. Sanchez-Gonzalez, A., Heess, N., Springenberg, J.T., Merel, J., Riedmiller, M., Hadsell, R., Battaglia, P.: Graph networks as learnable physics engines for inference and control. arXiv preprint arXiv:1806.01242 (2018)

20. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI magazine 29(3), 93–93 (2008)

21. Sønderby, C.K., Raiko, T., Maaløe, L., Sønderby, S.K., Winther, O.: How to train deep variational autoencoders and probabilistic ladder networks. In: 33rd International Conference on Machine Learning (ICML 2016) (2016)

22. Tang, L., Liu, H.: Leveraging social media networks for classification. Data Mining and Knowledge Discovery 23(3), 447–478 (2011)

23. Weisstein, E.W.: Jensen’s inequality. From MathWorld—A Wolfram Web Resource, https://mathworld.wolfram.com/JensensInequality.html

24. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596 (2019)

25. Yeung, S., Kannan, A., Dauphin, Y., Fei-Fei, L.: Tackling over-pruning in variational autoencoders. arXiv preprint arXiv:1706.03643 (2017)

26. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. arXiv preprint arXiv:1812.08434 (2018)