Chapter 5

EXPLORING CONVOLUTIONAL AUTO-ENCODERS

5.1 Introduction

Learning vector space embeddings of graph-structured or relational data are similar to embedding complex data into low-dimensional geometries. For an embedding method to be successful, this latent geometry must go beyond simple compression and provide an inductive bias for reasoning about the relationships between objects. After a latent geometry has been inferred, it is possible to represent the objects using their corresponding embeddings, and most importantly, these embeddings provide a natural feature space for these objects, which can then be used for network analysis models.

However, the proposed model seen in Chapter 4, were not suitable for situations where all network nodes are not available at the time of training, and network nodes are associated with attribute data. In such situations, “deep encoder" based techniques are more suitable as they map the nodes to their embedding vectors using a complex non-linear mapping function. The parameters used by the “deep encoder" based frameworks for mapping the nodes to their embedding vectors also consider attribute data associated with network nodes.

Deep learning architectures in the literature are based on Graph Convolutional Net-

works (G.C.N.). G.C.N. and its variants focus on information aggregation. A drawback of these models is the fully connected layer used in them increases the number of parameters. As the graph size increases, the number of parameters also increases, leading to slower convergence of gradient descent. Therefore the current inquiry proposes two auto-encoder architectures which modify existing techniques like Graph Auto-Encoder (G.A.E.) and Variational Graph Auto-Encoder (V.A.E.) by replacing the fully connected layers with 1-D convolutional ones.

The proposed architectures aim to maintain computational and storage efficiency, fewer parameters, localization, and applicability to inductive problems. Additionally, an equivalence is demonstrated between the aggregation and transformation operations performed in the hidden layers of an auto-encoder and laplacian smoothing operation.

5.1.1 Summary of contribution

- The main contribution in this chapter is to propose fully convolutional implementations of two standard G.C.N. architectures viz. Graph Auto-Encoder (G.A.E.) and Variational Graph Auto-Encoder (V.A.E.).
- Extensive experiments were performed on network data-sets to validate the proposed architectures viz. Graph Auto-Encoder with Convolutional layers (G.A.E.-F.C.) and Variational Graph Auto-Encoder with Convolutional layers (V.A.E.-F.C.) for latent space representation of networks.
- The advantages of these approaches are demonstrated over the standard Graph Auto-Encoder (G.A.E.) and Variational Graph Auto-Encoder (V.A.E.) frameworks by performing experiments on network data-sets.

The rest of the paper is organized as follows: In Section 5.2 a background of G.C.N. is provided, Section 5.3 describes the proposed method and Section 5.5 discusses the results. The conclusion of the inquiry is provided in Section 5.6.

5.2 Background of Graph Convolutional Networks

A graph $G(V, E)$ with V as the node-set and E as the dyad set has a binary adjacency matrix $A \in \mathbb{R}^{|V|*|V|}$ and a node feature matrix $X \in \mathbb{R}^{|V|*m}$ [36]. The node feature matrix is formed by stacking the m-dimension feature vector of each node horizontally. G.C.N. has the following steps:

1. The convolutional filter $\hat{A}$ is constructed from the new adjacency matrix $\tilde{A} = A+I$ and new degree matrix $\tilde{D} = D + I$ by the operation $\hat{A} = \tilde{D}^{\frac{-1}{2}}\tilde{A}\tilde{D}^{\frac{-1}{2}}$

2. A propagation rule for the graph convolution layer is defined as $H^{(1)} = (\hat{A}H^{(0)}\theta^{(0)})$. $H^{(1)}$ is matrix of activations of first layer, $H^{(0)} = X$, $\theta^{(0)}$ is a trainable weight matrix of layer 0 and is a non-linear activation function.

3. The convolved representations of the vertices $\hat{A}H^{(0)}$ are fed to a standard fully connected layer as given in Figure 5.1.

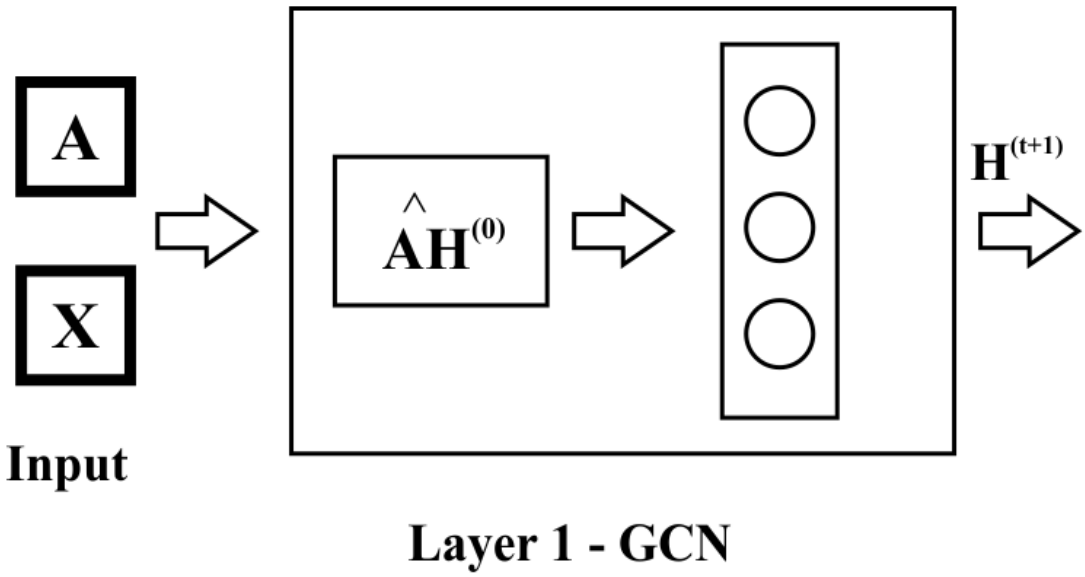


Figure 5.1: Graph Convolutional Network

5.3 Graph Auto-Encoder Network

Graph Auto-encoder proposed by T Kipf *et al.* takes as input the adjacency matrix A of the entire graph and the node feature matrix X. It outputs the reconstructed adjacency matrix $\hat{A}$ and the node embedding matrix Z. The loss function during learning is reconstruction loss which is calculated as $L = E_{q(Z|X,A)}[logp(A|Z)]$. An inner product function is used as the decoder $\hat{A} = \sigma(ZZ^T)$. As the encoder function minimizes the loss due to reconstruction error, the representations of nodes preserve first-order proximity.

Figure 5.2 provides the architecture of the encoder of the G.A.E. with 1-D fully convolutional (F.C.) layer in place of the fully connected (dense) layer. These F.C. encoder blocks are stacked to obtain the encoder of the G.A.E. The inner product decoder is kept the same.

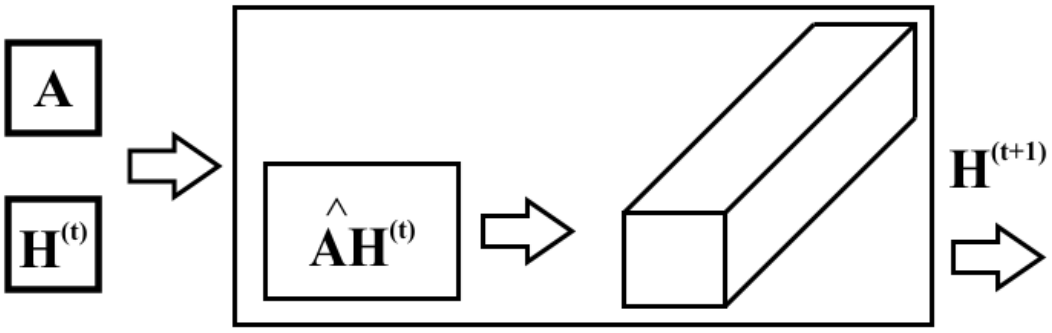


Figure 5.2: Architecture of Graph Auto-Encoder with Fully Convolutional layer (G.A.E.-F.C.)

5.4 Variational Graph Auto-Encoder Network

The Variational Graph Auto-Encoder (V.A.E.) proposed by T Kipf *et al.* [122] infers the latent variables Z by sampling from a normal distribution. The μ, σ^2 of this normal distribution is generated through a two-layer GCN as $\mu = GCN_\mu(X, A), \sigma^2 = GCN_\sigma(X, A), GCN(X, A) = \hat{A}ReLU(\hat{A}X\theta^{(0)})\theta^{(1)}$. Then for each input $x_i, q(Z_i|X, A) = N(Z_i|\mu_i, diag({\sigma_i}^2))$. The decoder is same as the inner product function of G.A.E. to

obtain the reconstructed adjacency matrix $A' = \sigma(Z.Z^T)$. The loss function has two parts as given in Eq 5.1: First part is the reconstruction loss to ensure reconstructed adjacency matrix A' is similar to input one A. The second part is KL-divergence to ensure distribution of latent variable is close to a normal distribution.

$$L = E_{q(Z|X,A)}[logp(A|Z)] - KL(q(Z|X,A)||p(Z)) \tag{5.1}$$

where, $p(Z) = \prod_i p(z_i) = \prod_i N(z_i|0, I)$

Similar to the 1-D convolutional Graph Auto-encoder, an F.C. encoder block is used to substitute the fully connected layer in the V.A.E. as given in Figure 5.3.

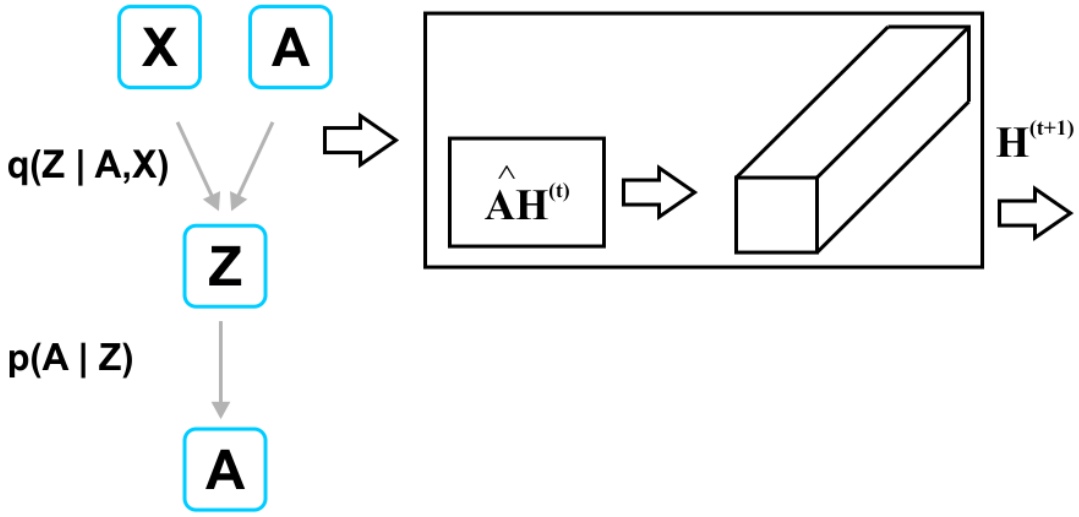


Figure 5.3: Architecture of Variational Graph Auto-encoder with Fully convolutional layer (VAE-FC)

5.4.1 Understanding neighbourhood-aggregation encoder algorithm of G.A.E., V.A.E., G.A.E.-F.C. and V.A.E.- F.C.

As given in Algorithm 5, the encoders of G.A.E., V.A.E., G.A.E.-F.C., and V.A.E.-F.C. use the neighbourhood aggregation strategy. There are two operations: AGGREGATE and COMBINE. AGGREGATE operation is common for G.A.E., V.A.E., G.A.E.-F.C., and

V.A.E.- F.C. The difference between convolutional implementations (G.A.E.-F.C. and V.A.E.- F.C) and other implementations (G.A.E. and V.A.E.) is in the COMBINE operation. The convolutional implementations use 1-D C.N.N. in the COMBINE operation wheres, the other implementations utilize a fully connected layer.

The complexity analysis, for an input graph having N nodes and feature vectors of length C, the adjacency matrix A will be in R^{N*N} and input feature matrix V^a will be of size R^{N*C}. For a given graph convolution layer, to obtain an output $V^{out} = R^{N*F}$, the time complexity is $O(N^2CF)$. A convolution operation with kernel size R^{p*p} in a conventional C.N.N., operating on the same input has a time complexity of $O(Np)$.

Algorithm 5: neighbourhood aggregation encoder algorithm of G.A.E., V.A.E., G.A.E.-F.C. and V.A.E.- F.C.. Adapted from [40]

Inputs: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\{\mathbf{W}^k, \forall k \in [1, K]\}$; non-linearity σ; differentiable aggregator functions $\{\text{AGGREGATE}_k, \forall k \in [1, K]\}$; neighbourhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Outpt : Vector embedding $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{COMBINE}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \text{NORMALIZE}(\mathbf{h}_v^k), \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

5.4.2 Advantages of fully convolutional layers

- Fixed parameters irrespective of the input size of the graph
- Fast convergence on gradient descent compared to fully connected layers
- Fewer parameters than fully connected layers

The two models proposed in Sections 5.3 and 5.4 are trained to learn embeddings using the procedure given in Algorithm 6.

Algorithm 6: Model Training and Fitting

Data: X, A

Result: Z

1 Initialization of encoder weights θ and bias b ;
2 Configure model to minimize Loss in Eq. 5.1;
3 Add activation function: rectified linear unit to encoder;
4 Create inner product decoder layer function;
5 Set epochs, batch size;
6 Optimize the loss function L using gradient descent;

5.4.3 Relation of auto-encoders with laplacian smoothing

Both the proposed auto-encoder models perform operations that are equivalent to a laplacian smoothing operation. Consider a curve with points p_i, p_{i-1}, p_{i+1}.. as shown in Figure 5.4 where laplacian smoothing is applied to bring each point closer to weighted average of its neighbours using Eq. 5.2 and 5.3.

$$p_i \leftarrow p_i + \frac{1}{2}L(p_i) \tag{5.2}$$

$$L(p_i) = \frac{p_{i+1} + P_{i-1}}{2} = p_i \tag{5.3}$$

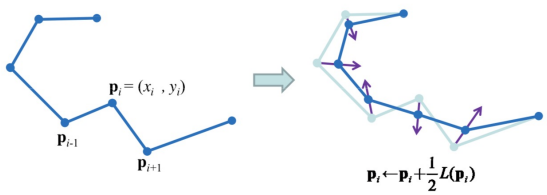


Figure 5.4: Example of laplacian smoothing where a curve is shown before and after the smoothing operation is applied [123]

For a graph $G = (V, E)$ with adjacency matrix A such that $a_{ij} = 1, |i - j| = 1$ and degree matrix $D = diag(d_1, d_2, .., d_n)$, the smoothing operation can be re-written as,

$$P \leftarrow (I - \frac{1}{2}L_{rw})P \tag{5.4}$$

where,

$$P = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \quad L_{\text{rw}} = \begin{bmatrix} 1 & -1 & & & \\ -\frac{1}{2} & 1 & -\frac{1}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1}{2} & 1 & -\frac{1}{2} \\ & & & -1 & 1 \end{bmatrix} \tag{5.5}$$

The matrix $L = D - A$ is the normalized graph Laplacian and has two versions $L_{sym} = D^{-1/2}LD^{-1/2}$ and $L_{rw} = D^{-1}L$. The laplacian smoothing operation is $P \leftarrow (I - \gamma L_{rw})P$ and $0 \leq \gamma \leq 1$ controls strength of smoothing. Setting $\gamma = 0$, laplacian smoothing becomes equivalent to non-linear mapping function (identity function) being learned by Auto-encoders. The Laplacian smoothing computes the local average of each vertex as its new representation. Since vertices in the same cluster tend to be densely connected, the smoothing makes their features similar, making the subsequent representation learning task amenable.

5.5 Experimental Study

5.5.1 Data-sets

Network data-sets with $|V| > 10^3$ were chosen for the evaluation of the proposed deep learning-based encoders technique viz. G.A.E.-F.C. and V.A.E.-F.C.

Table 5.1: Description of Network Data-sets with binary attributes

Description	**Blog-Net**	**Flickr-Net**	**Protein-Net**	**Wiki-net**	**Cora-net**	**Cite-net**
$\|V\|$	5196	7575	3890	4777	2708	3312
V^a	8189	12047	50	40	1434	3704
$\|E\|$	171743	239738	37845	54810	5429	4732
c	0.08	0.1	0.09	0.43	0.14	0.13

5.5.2 Encoder architectures

Encoder architectures of the Graph Auto-Encoder (G.A.E.), Variational Graph Auto-Encoder (V.A.E.), Graph Auto-Encoder with Convolutional layers (G.A.E.-F.C.) and Variational Graph Auto-Encoder with Convolutional layers (V.A.E.-F.C.) are described in Table 5.2, Table 5.3, Table 5.4 and Table 5.5 respectively. T Kipf *et al.* **??** implementation of G.A.E. and V.A.E is used. G.A.E.-F.C. and V.A.E.-F.C. are built using Keras package in Python [124].

Table 5.2: Dimensions of the dense layers in G.A.E encoder

layer name	Blog-Net	Cora-Net	Cite-Net	Flickr-Net	Protein-Net	Wiki-Net
X	5196*8189	2708*1434	3312*3704	7575*12047	3890*50	4777*40
A	5196*5196	2708*2708	3312*3312	7575*7575	3890*3890	4777*4777
dense_encode	8189*32	1434*32	3704*32	12047*32	50*32	40*32

Table 5.3: Dimensions of the dense layers in V.A.E. encoder

layer name	Blog-Net	Cora-Net	Cite-Net	Flickr-Net	Protein-Net	Wiki-Net
X	5196*8189	2708*1434	3312*3704	7575*12047	3890*50	4777*40
A	5196*5196	2708*2708	3312*3312	7575*7575	3890*3890	4777*4777
dense_encode	8189*64	1434*64	3704*64	12047*64	50*32	40*32
dense_encode	64*32	64*32	64*32	64*32	32*32	32*32

Table 5.4: Dimensions of the 1-D convolution layers in G.A.E-F.C. encoder

layer name	Blog-Net	Cora-Net	Cite-Net	Flickr-Net	Protein-Net	Wiki-Net
X	5196*8189	2708*1434	3312*3704	7575*12047	3890*50	4777*40
A	5196*5196	2708*2708	3312*3312	7575*7575	3890*3890	4777*4777
conv_encode	1x8158	1x1403	1x3673	1x12016	1x19	1x9
stride	1	1	1	1	1	1
height	1	1	1	1	1	1
padding	valid	valid	valid	valid	valid	valid

The architectures convert network data of Blog-Net, Cora-Net, Cite-Net, Flickr-Net, Protein-Net, and Wiki-Net from their original dimensions into vector space of dimension = 32.

Table 5.5: Dimensions of the 1-D convolution layer of V.A.E.-F.C. encoder

layer name	Blog-Net	Cora-Net	Cite-Net	Flickr-Net	Protein-Net	Wiki-Net
X	5196*8189	2708*1434	3312*3704	7575*12047	3890*50	4777*40
A	5196*5196	2708*2708	3312*3312	7575*7575	3890*3890	4777*4777
conv_encode	1x8126	1x1371	1x3641	1x11984	1x19	1x9
stride	1	1	1	1	1	1
height	1	1	1	1	1	1
padding	valid	valid	valid	valid	valid	valid
conv_encode	1x33	1*33	1x33	1x33	1*1	1x1
stride	1	1	1	1	1	1
height	1	1	1	1	1	1
padding	valid	valid	valid	valid	valid	valid

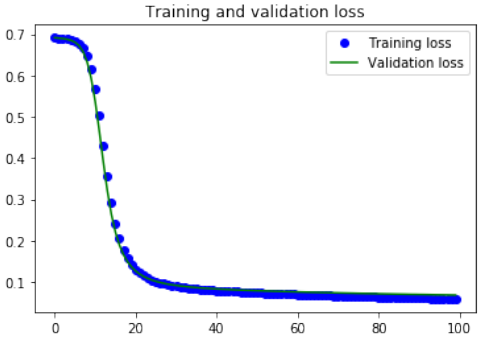
The quality of the vertex embeddings is measured using distance-based statistics obtained from clustering the representations. The parameters are separation index, widest within-cluster gap, average silhouette width, the average distance between clusters, and the Dunn index. Twenty iterations are performed on vertex embeddings for calculating each performance metric. The values of mean along with limits of two standard deviations $2SD$ of the mean (upper and lower bounds of the 95% confidence interval) are given for quantitative comparison. Additionally, the architectures are compared based on their parameters, time per iteration, and iterations required for convergence on the data-sets.

5.5.3 Results

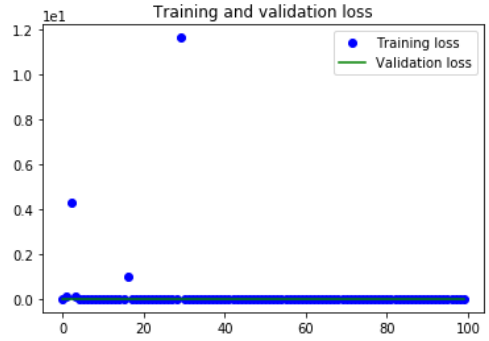
The two variants of the auto-encoder architecture are trained as per the Algorithm 6 to generate vector embeddings. The embeddings are then applied to node clustering to test the efficacy of the proposed models. Standard performance metrics described in the previous chapter viz. Separation index, Widest within-cluster gap, Average silhouette width, Average distance between clusters, and Dunn index are used to compare the clustering results. Results of a single iteration are given below.

Figure 5.5 shows binary cross-entropy loss (X-axis) for different epochs (Y-axis) on the training and test set of Blog-Net using Graph Auto-Encoder (G.A.E.) (Figure 5.5a),

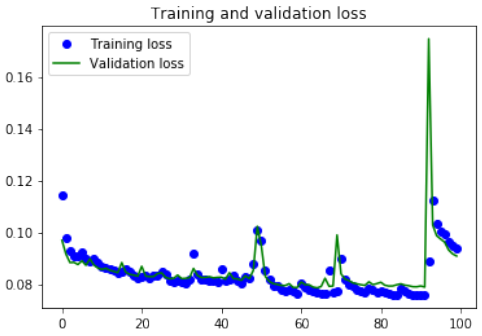
Variational Graph Auto-Encoder (V.A.E.) (Figure 5.5b), Graph Auto-Encoder with Convolutional layers (G.A.E.-F.C.) (Figure 5.5c) and Variational Graph Auto-Encoder with Convolutional layers (V.A.E.-F.C.) (Figure 5.5d).



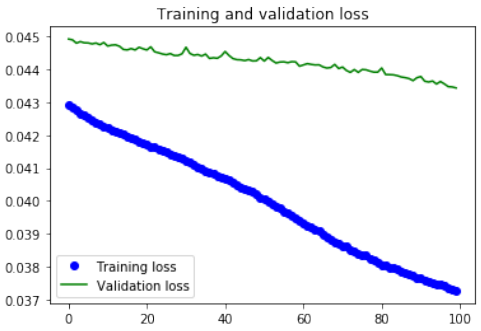
(a) Binary cross-entropy loss on training and test set of Blog-Net using G.A.E.



(b) Binary cross-entropy loss on training and test set of Blog-Net using V.A.E.



(c) Binary cross-entropy loss on training and test set of Blog-Net using G.A.E.-F.C.



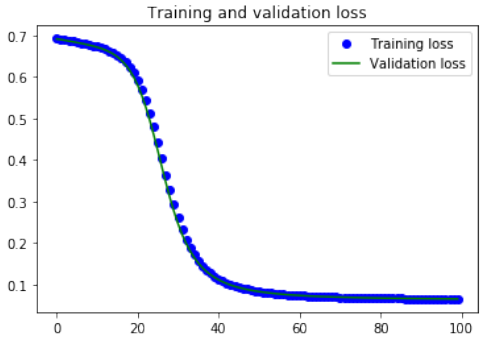
(d) Binary cross-entropy loss on training and test set of Blog-Net using V.A.E.-F.C.

Figure 5.5: Binary cross-entropy loss on training and test set of Blog-Net

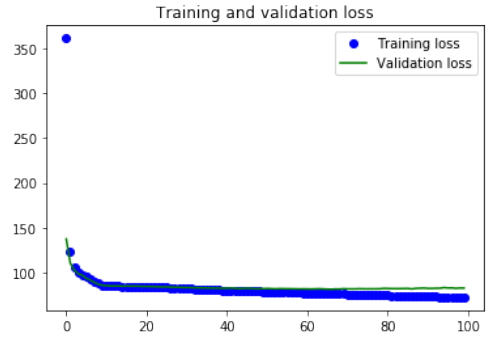
Figure 5.5a shows that loss on the test set converges to 0.012 after 96 iterations. Figure 5.5b shows that loss on the test set converges to 81 after 36 iterations. Figure 5.5c shows that loss on the test set is confined to range $0.08 - 0.1$ after 16 iterations. Figure 5.5d shows that loss on the test set decreases from $0.043 - 0.037$ with iterations.

Figure 5.6 shows binary cross-entropy loss (X-axis) for different epochs (Y-axis) on the training and test set of Cora-Net using Graph Auto-Encoder (G.A.E.) (Figure 5.6a), Variational Graph Auto-Encoder (V.A.E.) (Figure 5.6b), Graph Auto-Encoder with Convolutional layers (G.A.E.-F.C.) (Figure 5.6c) and Variational Graph Auto-Encoder

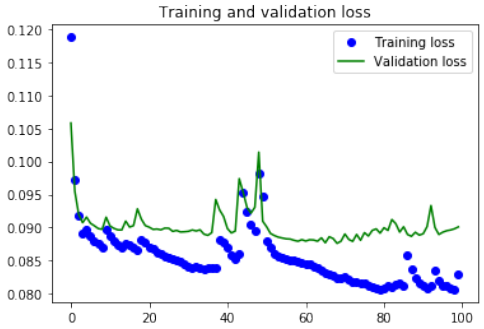
with Convolutional layers (V.A.E.-F.C.) (Figure 5.6d).



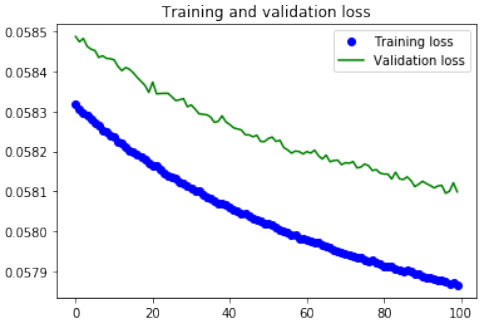
(a) Binary cross-entropy loss on training and test set of Cora-Net using G.A.E.



(b) Binary cross-entropy loss on training and test set of Cora-Net using V.A.E.



(c) Binary cross-entropy loss on training and test set of Cora-Net using G.A.E.-F.C.

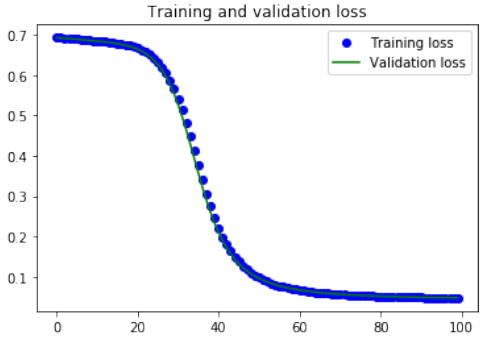


(d) Binary cross-entropy loss on training and test set of Cora-Net using V.A.E.-F.C.

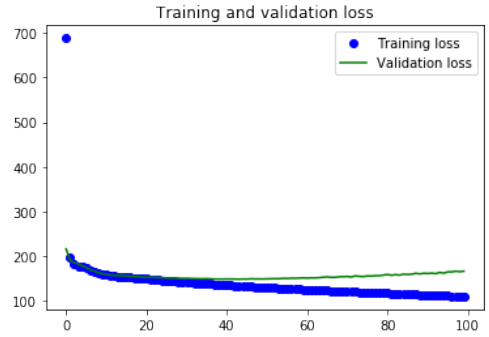
Figure 5.6: Binary cross-entropy loss on training and test set of Cora-Net

Figure 5.6a shows that loss on the test set converges to 0.012 after 42 iterations. Figure 5.6b shows that loss on the test set converges to 73 after 84 iterations. Figure 5.6c shows that loss on the test set is confined to range $0.09 - 0.8$ after 23 iterations. Figure 5.6d shows that loss on the test set decreases from $0.0583 - 0.0579$ with iterations.

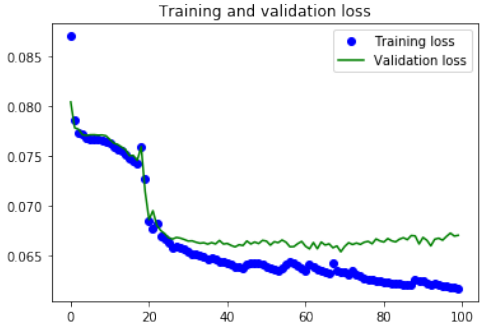
Figure 5.7 shows binary cross-entropy loss (X-axis) for different epochs (Y-axis) on the training and test set of Cite-Net using Graph Auto-Encoder (G.A.E.) (Figure 5.7a), Variational Graph Auto-Encoder (V.A.E.) (Figure 5.7b), Graph Auto-Encoder with Convolutional layers (G.A.E.-F.C.) (Figure 5.7c) and Variational Graph Auto-Encoder with Convolutional layers (V.A.E.-F.C.) (Figure 5.7d).



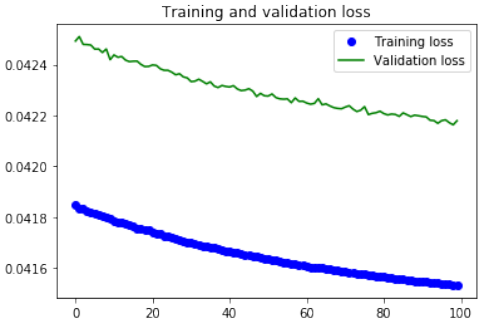
(a) Binary cross-entropy loss on training and test set of Cite-Net using G.A.E.



(b) Binary cross-entropy loss on training and test set of Cite-Net using V.A.E.



(c) Binary cross-entropy loss on training and test set of Cite-Net using G.A.E.-F.C.

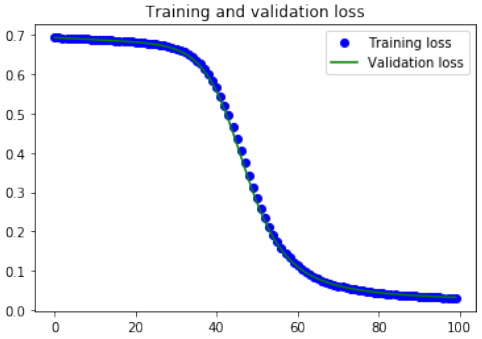


(d) Binary cross-entropy loss on training and test set of Cite-Net using V.A.E.-F.C.

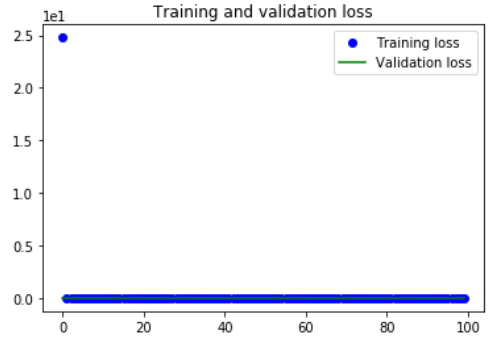
Figure 5.7: Binary cross-entropy loss on training and test set of Cite-Net

Figure 5.7a shows that loss on the test set converges to 0.008 after 92 iterations. Figure 5.7b shows that loss on the test set converges to 68 after 98 iterations. Figure 5.7c shows that loss on the test set is confined to range $0.08 - 0.65$ after 19 iterations. Figure 5.7d shows that loss on the test set decreases from $0.0424 - 0.0416$ with iterations.

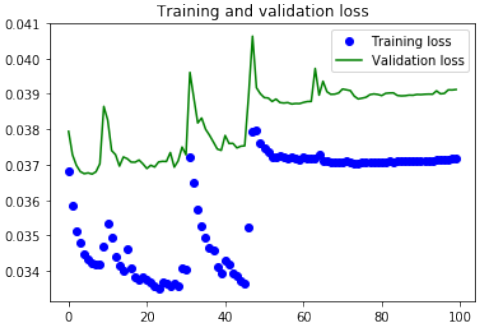
Figure 5.8 shows binary cross-entropy loss (X-axis) for different epochs (Y-axis) on the training and test set of Flickr-Net using Graph Auto-Encoder (G.A.E.) (Figure 5.8a), Variational Graph Auto-Encoder (V.A.E.) (Figure 5.8b), Graph Auto-Encoder with Fully Convolutional layers (G.A.E.-F.C.) (Figure 5.8c) and Variational Graph Auto-Encoder with Fully Convolutional layers (V.A.E.-F.C.) (Figure 5.8d).



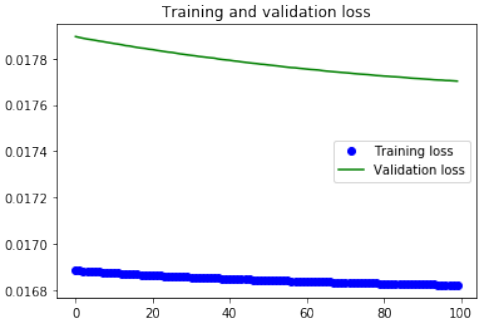
(a) Binary cross-entropy loss on training and test set of Flickr-Net using G.A.E.



(b) Binary cross-entropy loss on training and test set of Flickr-Net using V.A.E.



(c) Binary cross-entropy loss on training and test set of Flickr-Net using G.A.E.-F.C.

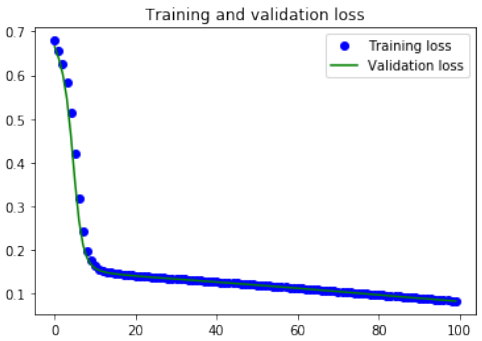


(d) Binary cross-entropy loss on training and test set of Flickr-Net using V.A.E.-F.C.

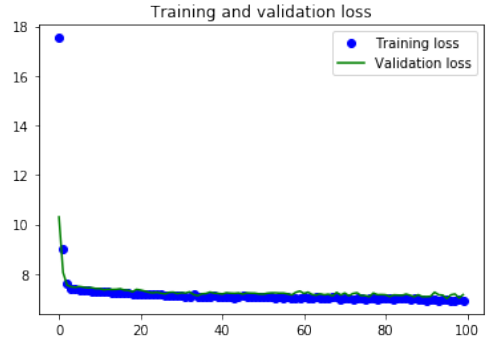
Figure 5.8: Binary cross-entropy loss on training and test set of Flickr-Net

Figure 5.8a shows that loss on the test set converges to 0.043 after 93 iterations. Figure 5.8b shows that loss on the test set converges to 629 after 98 iterations. Figure 5.8c shows that loss on the test set is confined to range $0.038 - 0.034$ after 23 iterations. Figure 5.8d shows that loss on the test set decreases from $0.0178 - 0.0176$ with iterations.

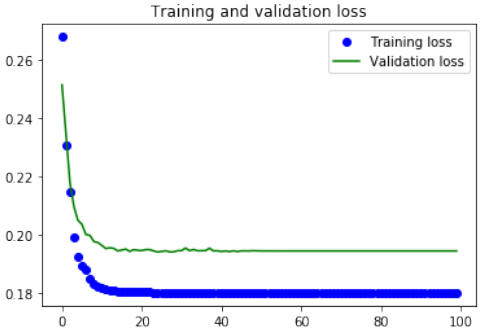
Figure 5.9 shows binary cross-entropy loss (X-axis) for different epochs (Y-axis) on the training and test set of Protein-Net using Graph Auto-Encoder (G.A.E.) (Figure 5.9a), Variational Graph Auto-Encoder (V.A.E.) (Figure 5.9b), Graph Auto-Encoder with Convolutional layers (G.A.E.-F.C.) (Figure 5.9c) and Variational Graph Auto-Encoder with Convolutional layers (V.A.E.-F.C.) (Figure 5.9d).



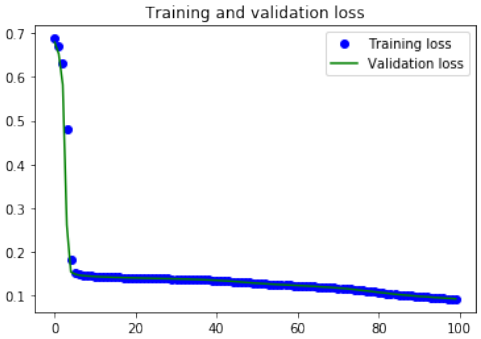
(a) Binary cross-entropy loss on training and test set of Protein-Net using G.A.E.



(b) Binary cross-entropy loss on training and test set of Protein-Net using V.A.E.



(c) Binary cross-entropy loss on training and test set of Protein-Net using G.A.E.-F.C.

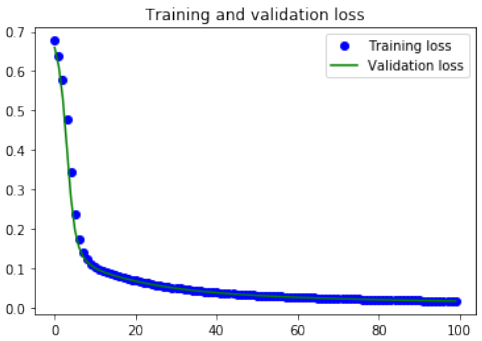


(d) Binary cross-entropy loss on training and test set of Protein-Net using V.A.E.-F.C.

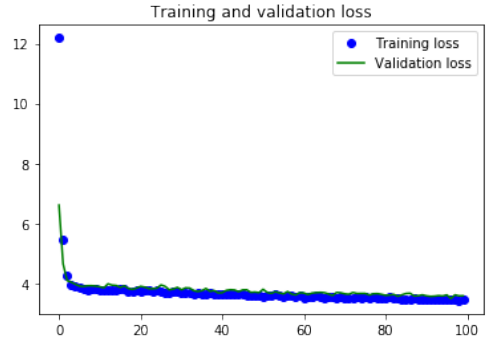
Figure 5.9: Binary cross-entropy loss on training and test set of Protein-Net

Figure 5.9a shows that loss on the test set converges to 0.008 after 94 iterations. Figure 5.9b shows that loss on the test set converges to 8 after 76 iterations. Figure 5.9c shows that loss on the test set is confined to range $0.21 - 0.18$ after 16 iterations. Figure 5.9d shows that loss on the test set decreases from $0.15 - 0.04$ with iterations.

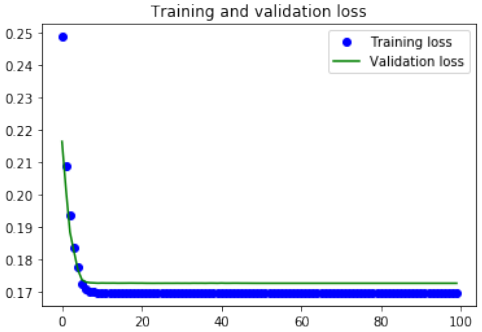
Figure 5.10 shows binary cross-entropy loss (X-axis) for different epochs (Y-axis) on the training and test set of Wiki-Net using Graph Auto-Encoder (G.A.E.) (Figure 5.10a), Variational Graph Auto-Encoder (V.A.E.) (Figure 5.10b), Graph Auto-Encoder with Convolutional layers (G.A.E.-F.C.) (Figure 5.10c) and Variational Graph Auto-Encoder with Convolutional layers (V.A.E.-F.C.) (Figure 5.10d).



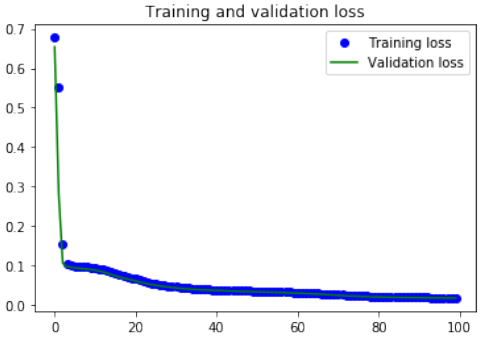
(a) Binary cross-entropy loss on training and test set of Wiki-Net using G.A.E.



(b) Binary cross-entropy loss on training and test set of Wiki-Net using V.A.E.



(c) Binary cross-entropy loss on training and test set of Wiki-Net using G.A.E.-F.C.



(d) Binary cross-entropy loss on training and test set of Wiki-Net using V.A.E.-F.C.

Figure 5.10: Binary cross-entropy loss on training and test set of Wiki-Net

Figure 5.10a shows that loss on the test set converges to 0.014 after 74 iterations. Figure 5.10b shows that loss on the test set converges to 4.2 after 36 iterations. Figure 5.10c shows that loss on the test set is confined to range 0.175 – 0.17 after 14 iterations. Figure 5.10d shows that loss on the test set decreases from 0.15 – 0.09 with iterations.

A comparison of the encoder blocks of the architectures is given in Table 5.6 (Number of parameters in the architectures) and Table 5.7 (Time required per iteration). As fully convolutional layers replace fully connected layers of G.A.E. and V.A.E., the number of parameters in G.A.E.-F.C. and V.A.E.-F.C. are reduced.

Table 5.6: Number of Parameters in encoder blocks of the auto-encoder architectures

Data-set	**G.A.E.**	**V.A.E.**	**G.A.E.-F.C.**	**V.A.E.-F.C**
Blog-Net	262048	526144	8158	8159
Cora-Net	45888	93824	1403	1404
Cite-Net	118528	239104	3673	3674
Flickr-Net	385504	773056	12016	12017
Protein-Net	1600	2624	19	20
Wiki-Net	1280	2304	9	10

Configuration of the system on which an experimental study was conducted is Intel(R) Core(T.M.) i5-6402P CPU@2.8GHz with four cores and 8GB DDR3 RAM. Table 5.7 gives the average time per iteration (in secs) of the auto-encoder architectures.

Table 5.7: Average time per iteration (in secs) of the auto-encoder architectures

Data-set	**G.A.E.**	**V.A.E.**	**G.A.E.-F.C.**	**V.A.E.-F.C**
Blog-Net	39 (±1.49)	59 (±1.84)	2.3 (±0.97)	3.5 (±0.67)
Cora-Net	24 (±1.31)	29 (±1.14)	1.23 (±0.64)	1.55 (±0.49)
Cite-Net	30 (±2.04)	37 (±2.11)	1.95 (±0.57)	2.16 (±0.4)
Flickr-Net	63 (±1.36)	89 (±1.91)	4.85 (±0.88)	5.76 (±0.76)
Protein-Net	11 (±0.67)	19 (±1.04)	0.89 (±0.11)	1.06 (±0.19)
Wiki-Net	16 (±0.58)	19 (±0.67)	0.86 (±0.09)	0.96 (±0.04)

The quality of the node embedding learned using the auto-encoder architectures were tested using distance-based statistics. Table 5.8 gives the results of parameter separation index for the data-sets. The L.S.R. obtained using G.A.E.-F.C. were higher (+) or lower (-) on separation index compared to the closest technique as mentioned: Wiki-Net (-0.5%), Cora-Net (-30%), Cite-Net (-0.26%), Blog-Net (-0.15%), Flickr-Net (-17%) and Protein-Net (22%). The L.S.R. obtained using V.A.E.-F.C. were higher (+) or lower (-) on separation index compared to the closest technique as mentioned: Wiki-Net (4%), Cora-Net (-32%), Cite-Net (5%), Blog-Net (23.5%), Flickr-Net (8%) and Protein-Net (38%).

Table 5.8: Comparison of G.A.E., V.A.E., G.A.E.-F.C., V.A.E.-F.C. on separation index

Data-set	**G.A.E.**	**V.A.E.**	**G.A.E.-F.C.**	**V.A.E.-F.C**
Blog-Net	4.24 (±0.76)	43.26 (±3.22)	4.29 (±0.71)	3.26 (±0.54)
Cora-Net	2.29 (±0.37)	37.19 (±3.92)	2.91 (±0.55)	2.93 (±0.57)
Cite-Net	6.18 (±1.16)	60.29 (±4.19)	6.19 (±0.64)	5.76 (±0.3)
Flickr-Net	7.16 (±1.39)	165.27 (±2.82)	8.16 (±0.67)	6.69 (±1.19)
Protein-Net	2.069 (±0.48)	8.16 (±0.81)	1.59 (±0.27)	0.97 (±0.08)
Wiki-Net	3.623 (±0.57)	13.29 (±2.09)	3.67 (±0.27)	2.86 (±0.19)

Table 5.9 gives the results of parameter widest within-cluster gap for the data-sets. The L.S.R. obtained using G.A.E.-F.C. were higher (+) or lower (-) on widest within-cluster gap compared to the closest technique as mentioned: Wiki-Net (11%), Cora-Net (-0.3%), Cite-Net (26%), Blog-Net (20%), Flickr-Net (21%) and Protein-Net (6%). The L.S.R. obtained using V.A.E.-F.C. were higher (+) or lower (-) on widest within-cluster gap compared to the closest technique as mentioned: Wiki-Net (8%), Cora-Net (-2%), Cite-Net (-5%), Blog-Net (55%), Flickr-Net (18%) and Protein-Net (3%).

Table 5.9: Comparison of G.A.E., V.A.E., G.A.E.-F.C., V.A.E.-F.C. on widest within-cluster gap

Data-set	**G.A.E.**	**V.A.E.**	**G.A.E.-F.C.**	**V.A.E.-F.C**
Blog-Net	2.49 (±0.67)	0.929 (±0.41)	3.16 (±0.34)	3.94 (±0.09)
Cora-Net	4.29 (±0.58)	2.549 (±0.73)	4.16 (±0.79)	4.27 (±0.2)
Cite-Net	2.59 (±0.54)	0.765 (±0.29)	2.94 (±0.61)	2.48 (±0.24)
Flickr-Net	4.19 (±0.91)	2.9 (±0.19)	4.93 (±0.52)	4.81 (±0.49)
Protein-Net	3.78 (±0.38)	3.4 (±0.18)	3.9 (±0.24)	3.98 (±0.42)
Wiki-Net	4.065 (±0.83)	3.952 (±0.64)	4.354 (±0.31)	4.289 (±0.67)

Table 5.10 gives the results of the parameter average silhouette width for the data-sets. The L.S.R. obtained using G.A.E.-F.C. were higher (+) or lower (-) on average silhouette width compared to the closest technique as mentioned: Wiki-Net (-4.9%), Cora-Net (2.3%), Cite-Net (-1.2%), Blog-Net (-2%), Flickr-Net (3.21%) and Protein-Net (4.6%). The L.S.R. obtained using V.A.E.-F.C. were higher (+) or lower (-) on average silhouette

width compared to the closest technique as mentioned: Wiki-Net (-2.8%), Cora-Net (2.3%), Cite-Net (-1.5%), Blog-Net (3.5%), Flickr-Net (6%) and Protein-Net (4.8%).

Table 5.10: Comparison of G.A.E., V.A.E., G.A.E.-F.C., V.A.E.-F.C. on average silhouette width

Data-set	**G.A.E.**	**V.A.E.**	**G.A.E.-F.C.**	**V.A.E.-F.C**
Blog-Net	0.19 (±0.0015)	-0.43 (±0.0048)	0.18 (±0.0091)	0.24 (±0.003)
Cora-Net	0.28 (±0.0048)	-0.19 (±0.0041)	0.31 (±0.005)	0.32 (±0.0065)
Cite-Net	0.31 (±0.0029)	-0.18 (±0.0015)	0.29 (±0.0049)	0.27 (±0.0034)
Flickr-Net	0.26 (±0.0048)	-0.64 (±0.0011)	0.34 (±0.007)	0.38 (±0.0019)
Protein-Net	0.37 (±0.0049)	0.16 (±0.0064)	0.43 (±0.0078)	0.44 (±0.0024)
Wiki-Net	0.49 (±0.0058)	0.18 (±0.0081)	0.38 (±0.0064)	0.45 (±0.002)

Table 5.11 gives the results of the parameter average distance between clusters for the data-sets. The L.S.R. obtained using G.A.E.-F.C. were higher (+) or lower (-) on the average distance between clusters compared to the closest technique as mentioned: Wiki-Net (6.4%), Cora-Net (14%), Cite-Net (-23%), Blog-Net (-6%), Flickr-Net (-23%) and Protein-Net (6.6%). The L.S.R. obtained using V.A.E.-F.C. were higher (+) or lower (-) on the average distance between clusters compared to the closest technique as mentioned: Wiki-Net (6.8%), Cora-Net (35%), Cite-Net (-15%), Blog-Net (5%), Flickr-Net (-16%) and Protein-Net (4.8%).

Table 5.11: Comparison of G.A.E., V.A.E., G.A.E.-F.C., V.A.E.-F.C. on average distance between clusters

Data-set	**G.A.E.**	**V.A.E.**	**G.A.E.-F.C.**	**V.A.E.-F.C**
Blog-Net	1.82 (±0.077)	0.49 (±0.059)	1.68 (±0.022)	1.92 (±0.082)
Cora-Net	3.82 (±0.089)	1.23 (±0.081)	4.26 (±0.034)	4.81 (±0.027)
Cite-Net	3.67 (±0.024)	0.96 (±0.088)	3.16 (±0.064)	3.24 (±0.084)
Flickr-Net	2.91 (±0.027)	1.02 (±0.018)	4.24 (±0.046)	3.91 (±0.044)
Protein-Net	1.29 (±0.12)	1.29 (±0.09)	1.84 (±0.18)	1.65 (±0.097)
Wiki-Net	1.21 (±0.091)	1.06 (±0.044)	1.82 (±0.038)	1.86 (±0.049)

Table 5.12 gives the results of parameter Dunn index for the data-sets. The L.S.R.

obtained using G.A.E.-F.C. were higher (+) or lower (-) on Dunn index compared to the closest technique as mentioned: Wiki-Net (-2%), Cora-Net (3%), Cite-Net (-2%), Blog-Net (-4%), Flickr-Net (-2%) and Protein-Net (4.6%). The L.S.R. obtained using V.A.E.-F.C. were higher (+) or lower (-) on Dunn index compared to the closest technique as mentioned: Wiki-Net (1.8%), Cora-Net (2%), Cite-Net (2.15%), Blog-Net (2.5%), Flickr-Net (-6.6%) and Protein-Net (3%).

Table 5.12: Comparison of G.A.E., V.A.E., G.A.E.-F.C., V.A.E.-F.C. on Dunn index

Data-set	**G.A.E.**	**V.A.E.**	**G.A.E.-F.C.**	**V.A.E.-F.C**
Blog-Net	0.23 (±0.019)	0.49 (±0.067)	0.27 (±0.018)	0.20 (±0.002)
Cora-Net	0.43 (±0.051)	0.61 (±0.037)	0.31 (±0.031)	0.37 (±0.014)
Cite-Net	0.19 (±0.046)	0.43 (±0.054)	0.21 (±0.028)	0.16 (±0.008)
Flickr-Net	0.18 (±0.009)	0.51 (±0.015)	0.21 (±0.092)	0.24 (±0.031)
Protein-Net	0.24 (±0.077)	0.26 (±0.034)	0.19 (±0.084)	0.21 (±0.02)
Wiki-Net	0.29 (±0.072)	0.34 (±0.048)	0.31 (±0.025)	0.27 (±0.061)

5.6 Discussion of Results and Summary

Deep learning architectures in the literature are based on Graph Convolutional Networks (G.C.N.). G.C.N. and its variants focus on information aggregation. A drawback of these models is the fully connected layer used in them increases the number of parameters. As the graph size increases, the number of parameters also increases. This leads to slower convergence of gradient descent. Therefore in auto-encoder architectures are proposed, which modify existing techniques like Graph Auto-Encoder (G.A.E.) and Variational Graph Auto-Encoder (V.A.E.) by replacing the fully connected layers with 1-D fully convolutional ones. The modified architectures are referred to as Graph Auto-Encoder with Fully Convolutional layers (G.A.E.-F.C.) and Variational Graph Auto-Encoder with Fully Convolutional layers (V.A.E.-F.C.).

- Figure 5.5 showed that the training and test loss on Blog-Net was 0.043 – 0.037 for V.A.E.-F.C. As G.A.E. and V.A.E.'s parameters were 262048 and 264096,

respectively, the training time was in the range of $39 - 59$ secs compared to that of the convolutional implementations ($2.3 - 3.5$ secs).

- A large number of parameters slowed the optimization. For the current experimental study, the number of epochs was limited to 100. The input size of Blog-Net is $X = 5196 * 8189. A = 5196 * 5196$ would require a larger number of epochs on the data. Between the G.A.E.-F.C. and V.A.E.-F.C., the former is a single-layered G.C.N. architecture, whereas the latter is a two-layered G.C.N. architecture. Consequently, the number of parameters of V.A.E.-F.C (8159) is higher than that of G.A.E.-F.C (8126).

- During the parameter training, the two-layered architecture saw a decrease in training and test loss with epochs compared to the single-layered architecture. This trend was also observed in Cora-Net ($X = 2708 * 1434, A = 2708 * 2708$), Cite-Net ($X = 3312 * 3704, A = 3312 * 3312$) and Flickr-Net ($X = 7575 * 12047, A = 7575 * 7575$).

- However, in Protein-Net ($X = 3890 * 50, A = 3890 * 3890$) and Wiki-Net ($X = 4777 * 40, A = 4777 * 4777$) as the input graphs had lesser dimensions, the auto-encoder architectures saw different trends. The convergence required less iterations (G.A.E. = 74, V.A.E. = 36, G.A.E.-F.C. = 14 and V.A.E.-F.C. = 36) and training and test loss was lower (G.A.E. = 0.014, V.A.E. = 4.2, G.A.E.-F.C. = 0.175-0.17 and V.A.E.-F.C. = 0.15-0.09).

Table 5.13 gives the results of the transitivity of the auto-encoder architectures on the data-sets. The L.S.R. obtained using G.A.E.-F.C. were higher (+) or lower (-) on transitivity compared to the closest technique as mentioned: Wiki-Net (-10%), Cora-Net (5%), Cite-Net (9%), Blog-Net (3%), Flickr-Net (8%) and Protein-Net (-7%). The L.S.R. obtained using V.A.E.-F.C. were higher (+) or lower (-) on transitivity compared to the closest technique as mentioned: Wiki-Net (-5%), Cora-Net (6%), Cite-Net (7%), Blog-Net (7%), Flickr-Net (3%) and Protein-Net (-2%).

With the advent of the digital transformation of society, data generation is occurring at an exponential rate.

Table 5.13: Comparison of G.A.E., V.A.E., G.A.E.-F.C., V.A.E.-F.C. on transitivity

Data-set	**Actual** c	**G.A.E.**	**V.A.E.**	**G.A.E.-F.C.**	**V.A.E.-F.C**
Blog-Net	0.08	0.19 (±0.032)	0.41 (±0.091)	0.16 (±0.082)	0.11 (±0.032)
Cora-Net	0.09	0.13 (±0.075)	0.56 (±0.064)	0.08 (±0.009)	0.07 (±0.07)
Cite-Net	0.13	0.18 (±0.064)	0.43 (±0.032)	0.09 (±0.064)	0.11 (±0.038)
Flickr-Net	0.1	0.16 (±0.093)	0.34 (±0.082)	0.08 (±0.014)	0.13 (±0.027)
Protein-Net	0.09	0.06 (±0.034)	0.16 (±0.051)	0.13 (±0.028)	0.08 (±0.004)
Wiki-Net	0.43	0.37 (±0.035)	0.47 (±0.064)	0.37 (±0.023)	0.42 (±0.008)

Network data does not have all nodes available at the time of training, and also network nodes are associated with attribute data. Representation learning for such network data is not feasible using proposed techniques in Chapter 4. Hence, two deep learning-based graph encoders were proposed viz. viz. Graph Auto-Encoder with Convolutional layers (G.A.E.-F.C.) and Variational Graph Auto-Encoder with Convolutional layers (V.A.E.-F.C.). It was observed that compared to G.A.E. and V.A.E., the proposed architectures performed better at representation learning.

Such architectures were also able to learn representations for data-points not observed in the training phase. This feature makes them more suitable for network data, which may have attributed nodes. The operation of aggregation and transformation performed in the hidden layer of a graph neural network is equivalent to laplacian smoothing operation. Hence, Auto-encoder architectures learn representations such that data-points associated with similar characteristics in the high dimension space are located at proximity in the latent space. Experiments performed on network data-sets experimentally validated this theoretical insight.

The deep graph neural network architectures in the literature and proposed in Chapter 5 are two layers deep. Therefore, these G.C.N. based architectures have a small depth compared to deep learning networks in the image and natural language processing. Thus, to improve training accuracy and reduce the loss on data, a much deeper architecture would be required. Hence, in Chapter 6 an investigation of the effects of the addition of a residual connection to the Graph Convolutional Network architecture is presented.

5.7 Publications

Discussions and results in this chapter are published in:

5.7.1 Journals

1. Nerurkar, P., Chandane, M. and Bhirud, S., (2019). Exploring convolutional auto-encoders for representation learning on networks. Computer Science, 20(3), pp. 1-13. (AGH University, Poland)