TiDL Assignment-2

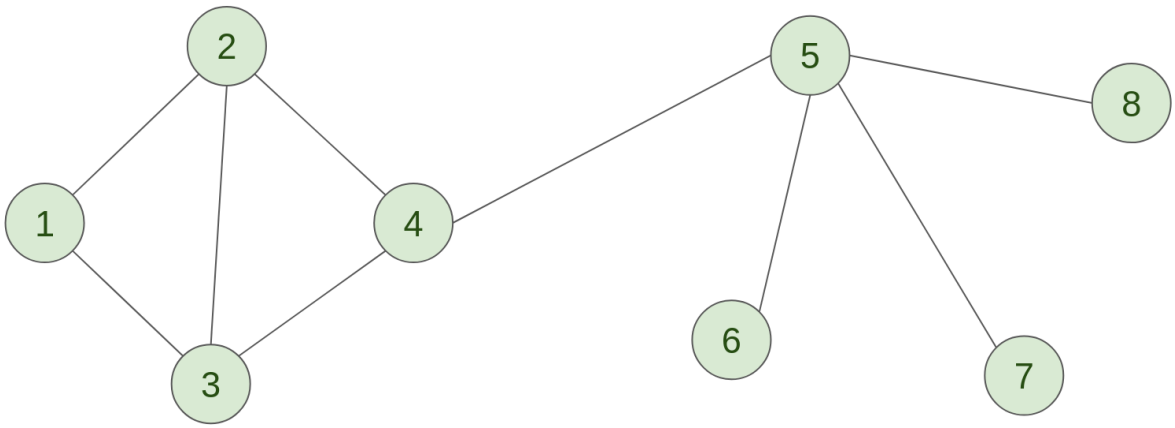
2018114018

Q1.

Part 1.

Graph

Given below is the 8-node graph we will be running the Random Walk approaches on:



Loss Function and Training Formulation

We have the loss function:

$$L = \sum_{u \in V} \sum_{v \in N_R(u)} -log\left(\frac{e^{Z_u^T Z_v}}{\sum_{k \in V} e^{Z_u^T Z_k}}\right)$$

where:

- V: All nodes in G
- N_R(u): Neighbourhood of u (for each of computation, taken as the nodes obtained in a random walk from u)

We then compute the partial derivative of the loss w.r.t Z_u:

$$\frac{\partial L}{\partial z_u} = -\sum_{v \in N_R(u)} \left(z_v - \left(\frac{\sum_k e^{z_u^T z_k} z_k}{\sum_k e^{z_u^T z_k}}\right)\right)$$

$$= \sum_{v \in N_R(u)} \left(\frac{\sum_k e^{z_u^T z_k} z_k}{\sum_k e^{z_u^T z_k}} - z_v\right)$$

We can then use the above expression to obtain the update equation for the gradient descent:

$$z_u^{(k+1)} = z_u^{(k)} - \alpha \frac{\partial L}{\partial z_u^k}$$

DeepWalk

Given below are the initial embeddings (dim=3) for each graph-node:

```
1: [3. 1. 4.]
2: [1. 5. 9.]
3: [2. 6. 5.]
4: [3. 5. 8.]
5: [9. 7. 9.]
6: [3. 2. 3.]
7: [8. 4. 6.]
8: [2. 6. 4.]
```

- **Iteration 1**

  We randomly generate random walks for each node (walk length=4)

  ```
  1: [3, 2, 4, 5]
  2: [1, 3, 1, 3]
  3: [2, 4, 3, 2]
  4: [2, 4, 3, 1]
  5: [6, 5, 6, 5]
  6: [5, 7, 5, 4]
  7: [5, 6, 5, 8]
  8: [5, 7, 5, 7]
  ```

  Now we compute the partial derivatives at the matrix-level:

  ```
  del_L / del_Z_0

  [[21.000 5.000  5.000 ]
   [30.200 15.000 19.000]
   [35.040 10.000 8.800 ]
   [35.624 14.000 13.280]
   [12.000 10.000 12.000]
   [8.163  4.399  2.927 ]
   [11.416 4.440  8.893 ]
   [1.883  4.888  5.379 ]]
  ```

  Which is then used to update Z_0 to Z_1:

```
[[0.900  0.500 3.500]
 [-2.020 3.500 7.100]
 [-1.504 5.000 4.120]
 [-0.562 3.600 6.672]
 [7.800  6.000 7.800]
 [2.184  1.560 2.707]
 [6.858  3.556 5.111]
 [1.812  5.511 3.462]]
```

- **Iteration 2**

  We randomly generate random walks for each node (walk length=4)

```
1: [2, 3, 4, 2]
2: [4, 5, 4, 5]
3: [2, 4, 3, 1]
4: [3, 2, 4, 5]
5: [8, 5, 7, 5]
6: [5, 4, 5, 8]
7: [5, 8, 5, 4]
8: [5, 8, 5, 6]
```

  Now we compute the partial derivatives at the matrix-level:

```
del_L / del_Z_0

[[37.306  8.400  6.208 ]
 [-22.448 -5.177 -0.544]
 [28.356  9.754  9.694 ]
 [28.054  6.352  6.421 ]
 [6.930   2.933  7.027 ]
 [15.770  2.937  4.702 ]
 [15.770  2.937  4.703 ]
 [11.796  4.636  8.495 ]]
```

  Which is then used to update Z_1 to Z_2:

```
[[-2.831 -0.340 2.879]
 [0.225  4.018  7.154]
 [-4.340 4.025  3.151]
 [-3.368 2.965  6.030]
 [7.107  5.707  7.097]
 [0.607  1.266  2.237]
 [5.281  3.262  4.640]
 [0.632  5.048  2.613]]
```

Node2Vec

Although the overall loss function remains the same as in DeepWalk, the Random Walks are no longer arbitrarily random, but contingent on a node's neighbourhood.

Given below are the initial embeddings (dim=3) for each graph-node (Note that these are the same as the ones used for DeepWalk to aid in later comparison):

```
1: [3. 1. 4.]
2: [1. 5. 9.]
3: [2. 6. 5.]
4: [3. 5. 8.]
5: [9. 7. 9.]
6: [3. 2. 3.]
7: [8. 4. 6.]
8: [2. 6. 4.]
```

- **Iteration 1**

  We randomly generate random walks for each node (walk length=4)

  ```
  1: [3, 2, 4, 5]
  2: [3, 4, 2, 3]
  3: [2, 1, 3, 2]
  4: [5, 6, 5, 7]
  5: [7, 5, 7, 5]
  6: [5, 6, 5, 4]
  7: [5, 6, 5, 7]
  8: [5, 4, 2, 3]
  ```

  Now we compute the partial derivatives at the matrix-level:

  ```
  del_L / del_Z_0

  [[21.000 5.000  5.000 ]
   [28.000 6.000  9.000 ]
   [36.700 12.700 11.300]
   [7.000  8.000  9.000 ]
   [2.000  6.000  6.000 ]
   [12.300 6.600  6.700 ]
   [7.830  7.460  8.470 ]
   [27.570 5.870  6.130 ]]
  ```

  Which is then used to update Z_0 to Z_1:

  ```
  [[0.900  0.500 3.500]
   [-1.800 4.400 8.100]
   [-1.670 4.730 3.870]
   [2.300  4.200 7.100]
   [8.800  6.400 8.400]
   [1.770  1.340 2.330]
   [7.217  3.254 5.153]
   [-0.757 5.413 3.387]]
  ```

- **Iteration 2**

  We randomly generate random walks for each node (walk length=4)

```
1: [2, 3, 4, 2]
2: [1, 3, 4, 2]
3: [1, 2, 1, 3]
4: [5, 8, 5, 7]
5: [6, 5, 6, 5]
6: [5, 4, 3, 1]
7: [5, 4, 2, 3]
8: [5, 7, 5, 7]
```

Now we compute the partial derivatives at the matrix-level:

```
del_L / del_Z_0

[[38.169 7.870  6.430 ]
 [-3.097 4.560  10.473]
 [19.999 11.716 13.784]
 [11.140 4.133  8.260 ]
 [14.060 10.120 12.140]
 [27.583 9.103  9.932 ]
 [26.156 4.875  5.740 ]
 [-0.719 4.223  5.100 ]]
```

Which is then used to update Z_1 to Z_2:

```
[[-2.917 -0.287 2.857]
 [-1.490 3.944  7.053]
 [-3.670 3.558  2.492]
 [1.186  3.787  6.274]
 [7.394  5.388  7.186]
 [-0.988 0.430  1.337]
 [4.601  2.767  4.579]
 [-0.685 4.991  2.877]]
```

Part 2.

For the sake of structural analysis, we note the following:

- Nodes 2 & 3 are strcurally highly similar (Both are attached to Nodes 1 & 4, and an edge joining them)
- Nodes 6, 7, and 8 are structrally highly similar (all have a simgle connection to Node 5)

This would suggest that the algorithm which is able to bring the above pairs and triplets closer in terms of their node-embeddgings fairs better.

[Note that the similarities will not be highly pronounced, as the weights are NOT being updated after each iteration.]

- Nodes 2 & 3:
  - DeepWalk: 0.6856
  - Node2Vec: 0.7934

This shows that Node2Vec has been able to capture the said similarity better.

Node2Vec's random walk policy has helped here, by providing structurally more relevant node sequences compared to DeepWalk.

Q2.

Part 1.

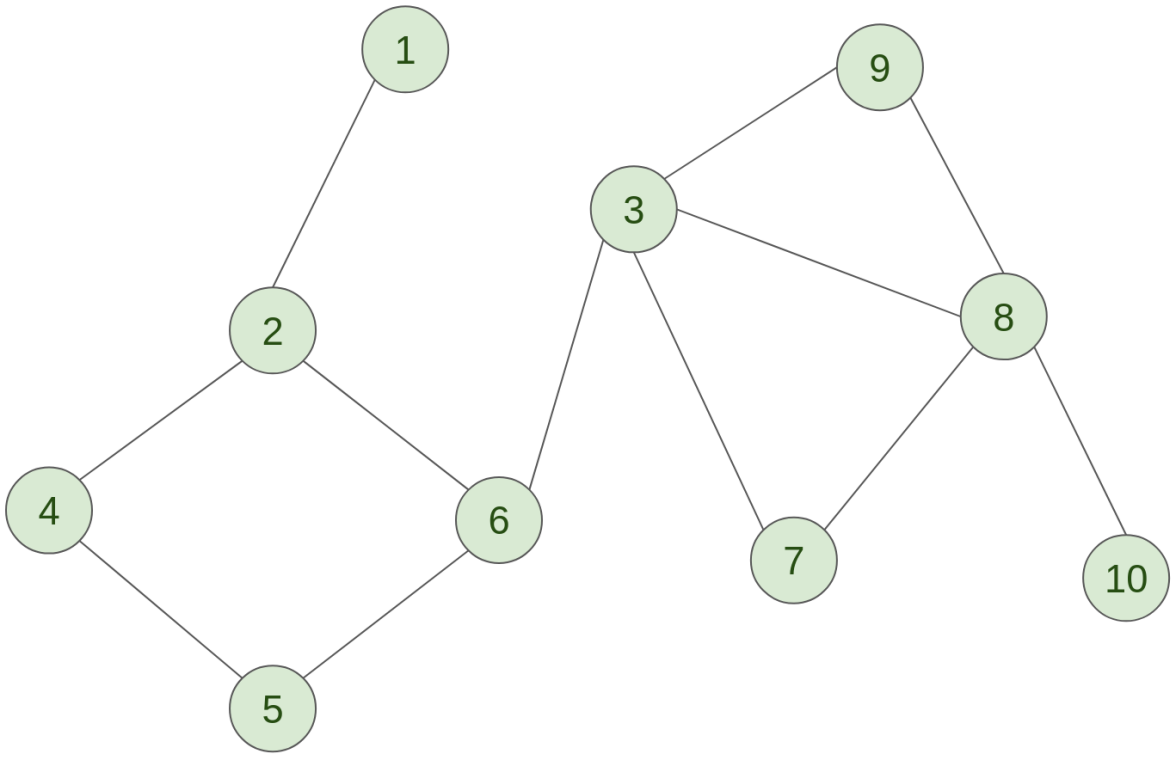
For this question, we will be running three versions of a simple **Graph Convolution Network (GCN)** with *1, 2, and 3 GCN layers* respectively.

Assumptions:

1. As there does not exist an end-to-end code to run the GNN models, we will be omitting the backpropogation step. This suggests that for each iteration, we shall be assuming a new distribution for the weight matrix (which is not affected by the outputs of any previous iterations).
2. The initial graph-node represntations have been assumed to be one-hot vectors for the sake of simplificty.
3. Applying L2 Norm after every layer has been omitted for computational ease as well as better inter-layer and inter-iteration comparison.

Graph

Given below is the 10-node graph we will be training our GCN models on:



Run 1 (GCN, Layers = 1)

- Initialization

```
 Raw A (Adjacency Matrix)
==========================

[[0 1 0 0 0 0 0 0 0 0]
 [1 0 0 1 0 1 0 0 0 0]
 [0 0 0 0 0 1 1 1 1 0]
 [0 1 0 0 1 0 0 0 0 0]
 [0 0 0 1 0 1 0 0 0 0]
 [0 1 1 0 1 0 0 0 0 0]
 [0 0 1 0 0 0 0 1 0 0]
 [0 0 1 0 0 0 1 0 1 1]
 [0 0 1 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0 0]]

 A cap (A = A + I)
===============

[[1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 0. 1. 0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 1. 1. 1. 1. 0.]
 [0. 1. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 1. 1. 0. 1. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 1. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 1. 0. 0. 0. 0. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 1.]]

 D (Degree Matrix)
===================

[[3. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 4. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 5. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 3. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 3. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 4. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 3. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 5. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 3. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 2.]]

 X (Node Representation)
========================

[[0.5  0.49 0.2  0.75 0.14 0.51 0.65 0.09 0.25 0.82]
 [0.78 0.47 0.54 0.76 0.78 0.88 0.04 0.88 1.   0.98]
 [0.84 0.74 0.11 0.17 0.63 0.07 0.7  0.12 0.92 0.97]
 [0.56 0.1  0.03 0.47 0.85 0.66 0.95 0.3  0.48 0.87]
 [0.49 0.91 0.84 0.78 0.82 0.45 0.74 0.2  0.86 0.73]
 [0.96 0.82 0.48 0.84 0.99 0.75 0.98 0.67 0.44 0.03]
 [0.32 0.04 0.73 0.34 0.69 0.5  0.99 0.04 0.27 0.08]
```

```
 [0.74 1.   0.67 0.92 0.23 0.62 0.69 0.88 0.7  0.6 ]
 [0.4  0.53 0.83 0.84 0.46 0.18 0.26 0.95 0.96 0.07]
 [0.24 0.4  0.76 0.09 0.63 0.21 0.61 0.52 0.13 0.79]]
```

- **Iteration 1**

```
 W_0 (Weight Matrix)
====================

[[0.85 0.92 0.17 0.87 0.19 0.36 0.23 0.33 0.75 0.9 ]
 [0.42 0.36 0.82 0.14 0.17 0.77 0.   0.37 0.43 0.33]
 [0.12 0.37 0.24 0.09 0.47 0.19 0.91 0.56 0.73 0.15]
 [0.4  0.14 0.74 0.1  0.59 0.63 0.26 0.56 0.22 0.56]
 [0.65 0.51 0.87 0.74 0.36 0.93 0.09 0.88 0.52 0.29]
 [0.68 0.46 0.66 0.98 0.17 0.71 0.1  0.71 0.77 0.2 ]
 [0.95 0.52 0.1  0.28 0.47 0.09 0.19 0.43 0.5  0.11]
 [0.81 0.1  0.94 0.35 0.08 0.58 0.6  0.45 0.78 0.66]
 [0.68 0.47 0.66 0.49 0.07 0.16 0.36 0.35 0.9  0.19]
 [0.59 0.17 0.52 0.46 0.33 0.57 0.44 0.39 0.23 0.63]]
```

Computing for Layer-1

$$X^{(1)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(0)} * W^{(0)}\right)$$

```
X_1

[[3.82 2.41 3.74 3.02 1.71 3.27 1.96 3.12 3.56 2.75]
 [3.83 2.46 3.39 2.9  1.75 3.07 1.67 3.04 3.35 2.5 ]
 [3.59 2.42 3.2  2.49 1.69 2.75 1.77 2.86 3.39 2.27]
 [4.01 2.59 3.72 3.06 1.87 3.28 1.93 3.29 3.64 2.6 ]
 [4.06 2.71 3.55 2.96 1.95 3.21 1.75 3.3  3.59 2.46]
 [4.09 2.76 3.76 3.04 1.87 3.31 1.9  3.28 3.77 2.67]
 [3.44 2.33 2.88 2.39 1.62 2.52 1.64 2.65 3.12 2.12]
 [3.21 2.12 2.88 2.19 1.55 2.46 1.74 2.58 3.05 2.05]
 [3.64 2.37 3.42 2.44 1.64 2.81 1.92 2.8  3.48 2.47]
 [3.44 2.17 3.21 2.31 1.69 2.79 1.97 2.79 3.24 2.33]]
```

- **Iteration 2**

```
 W_1 (Weight Matrix)
====================

[[0.51 0.24 0.03 0.07 0.91 0.26 0.14 0.72 0.5  0.2 ]
 [0.42 0.55 0.07 0.75 0.08 0.81 0.71 0.29 0.6  0.06]
 [0.76 0.64 0.48 0.61 0.09 0.8  0.47 0.85 0.77 0.94]
 [0.51 0.42 0.56 0.77 0.95 0.93 0.71 0.81 1.   0.04]
 [0.12 0.09 0.85 0.29 0.15 0.62 0.18 0.44 0.37 0.7 ]
 [0.67 0.72 0.67 0.64 0.45 0.26 0.83 0.83 0.8  0.04]
 [0.9  0.32 0.71 0.02 0.76 0.29 0.24 0.76 0.46 0.08]
 [0.84 0.76 0.91 0.37 0.78 0.99 0.08 0.75 0.6  0.09]
 [0.61 0.43 0.68 0.97 0.94 0.26 0.8  0.64 0.29 0.47]
 [0.92 0.41 0.95 0.72 0.88 0.34 0.15 0.8  0.24 0.69]]
```

Computing for Layer-1

$$X^{(1)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(0)} * W^{(1)}\right)$$

```
X_1

[[18.06 13.58 16.05 15.37 17.67 15.71 12.74 20.12 16.43  9.46]
 [18.69 14.06 16.61 15.91 18.25 16.26 13.21 20.81 17.    9.8 ]
 [16.78 12.56 14.87 14.23 16.39 14.53 11.82 18.61 15.12  8.84]
 [18.42 13.91 16.39 15.72 18.05 16.17 13.06 20.56 16.85  9.64]
 [18.95 14.31 16.85 16.18 18.53 16.61 13.45 21.12 17.31  9.93]
 [18.06 13.62 16.04 15.42 17.68 15.81 12.82 20.12 16.46  9.48]
 [15.8  11.83 14.02 13.39 15.48 13.71 11.13 17.53 14.24  8.3 ]
 [16.24 12.12 14.41 13.69 15.84 13.98 11.37 17.99 14.56  8.57]
 [16.35 12.21 14.5  13.81 15.94 14.08 11.47 18.11 14.66  8.64]
 [15.84 11.79 14.11 13.25 15.41 13.57 11.01 17.53 14.14  8.35]]
```

- **Iteration 3**

```
 W_2 (Weight Matrix)
====================

[[0.76 0.56 0.01 0.63 0.35 0.83 0.44 0.45 0.63 0.98]
 [0.78 0.83 0.38 0.03 0.16 0.68 0.2  0.49 0.9  0.56]
 [0.87 0.73 0.68 0.74 0.73 1.   0.1  0.2  0.09 0.18]
 [0.02 0.03 0.04 0.24 0.25 0.8  0.65 0.33 0.2  0.2 ]
 [0.66 0.88 0.28 0.49 0.5  0.45 0.82 0.27 0.35 0.4 ]
 [0.53 0.02 0.1  0.78 0.19 0.55 0.9  0.37 0.67 0.56]
 [0.39 0.48 0.25 0.64 0.91 0.86 0.09 0.05 0.46 0.84]
 [0.95 0.13 0.61 0.02 0.3  0.08 0.56 0.11 0.59 0.31]
 [0.45 0.39 0.94 0.45 0.79 0.96 0.28 0.53 0.68 0.04]
 [0.19 0.12 0.06 0.97 0.56 0.55 0.55 0.77 0.27 0.56]]
```

Computing for Layer-1

$$X^{(1)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(0)} * W^{(2)}\right)$$

```
X_1

[[ 94.2   67.13  56.35  75.12  73.38 104.85  74.92  53.78  78.01  71.65]
 [ 93.43  66.57  55.89  74.52  72.78 104.01  74.33  53.35  77.39  71.07]
 [ 84.63  60.33  50.54  67.55  65.91  94.16  67.32  48.32  70.02  64.43]
 [ 95.42  67.98  57.09  76.11  74.33 106.23  75.91  54.48  79.05  72.59]
 [ 94.39  67.25  56.48  75.31  73.54 105.11  75.12  53.91  78.22  71.82]
 [ 92.39  65.84  55.24  73.72  71.98 102.86  73.51  52.76  76.52  70.32]
 [ 82.67  58.94  49.35  65.98  64.37  91.96  65.75  47.19  68.38  62.94]
 [ 82.23  58.63  49.08  65.62  64.02  91.44  65.38  46.94  67.99  62.59]
 [ 83.53  59.55  49.85  66.67  65.04  92.91  66.43  47.69  69.08  63.6 ]
 [ 81.3   57.96  48.51  64.85  63.26  90.34  64.57  46.38  67.15
61.84]]
```

Run 2 (GCN, Layers = 2)

- Initialization

```
 Raw A (Adjacency Matrix)
==========================

[[0 1 0 0 0 0 0 0 0 0]
 [1 0 0 1 0 1 0 0 0 0]
 [0 0 0 0 0 1 1 1 1 0]
 [0 1 0 0 1 0 0 0 0 0]
 [0 0 0 1 0 1 0 0 0 0]
 [0 1 1 0 1 0 0 0 0 0]
 [0 0 1 0 0 0 0 1 0 0]
 [0 0 1 0 0 0 1 0 1 1]
 [0 0 1 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0 0]]

 A cap (A = A + I)
===============

[[1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 0. 1. 0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 1. 1. 1. 1. 0.]
 [0. 1. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 1. 1. 0. 1. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 1. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 1. 0. 0. 0. 0. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 1.]]

 D (Degree Matrix)
===================

[[3. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 4. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 5. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 3. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 3. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 4. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 3. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 5. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 3. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 2.]]

 X (Node Representation)
========================

[[0.5  0.49 0.2  0.75 0.14 0.51 0.65 0.09 0.25 0.82]
 [0.78 0.47 0.54 0.76 0.78 0.88 0.04 0.88 1.   0.98]
 [0.84 0.74 0.11 0.17 0.63 0.07 0.7  0.12 0.92 0.97]
 [0.56 0.1  0.03 0.47 0.85 0.66 0.95 0.3  0.48 0.87]
 [0.49 0.91 0.84 0.78 0.82 0.45 0.74 0.2  0.86 0.73]
 [0.96 0.82 0.48 0.84 0.99 0.75 0.98 0.67 0.44 0.03]
```

```
 [0.32 0.04 0.73 0.34 0.69 0.5  0.99 0.04 0.27 0.08]
 [0.74 1.   0.67 0.92 0.23 0.62 0.69 0.88 0.7  0.6 ]
 [0.4  0.53 0.83 0.84 0.46 0.18 0.26 0.95 0.96 0.07]
 [0.24 0.4  0.76 0.09 0.63 0.21 0.61 0.52 0.13 0.79]]
```

- **Iteration 1**

```
 W_0 (Weight Matrix)
==================

[[0.85 0.92 0.17 0.87 0.19 0.36 0.23 0.33 0.75 0.9 ]
 [0.42 0.36 0.82 0.14 0.17 0.77 0.   0.37 0.43 0.33]
 [0.12 0.37 0.24 0.09 0.47 0.19 0.91 0.56 0.73 0.15]
 [0.4  0.14 0.74 0.1  0.59 0.63 0.26 0.56 0.22 0.56]
 [0.65 0.51 0.87 0.74 0.36 0.93 0.09 0.88 0.52 0.29]
 [0.68 0.46 0.66 0.98 0.17 0.71 0.1  0.71 0.77 0.2 ]
 [0.95 0.52 0.1  0.28 0.47 0.09 0.19 0.43 0.5  0.11]
 [0.81 0.1  0.94 0.35 0.08 0.58 0.6  0.45 0.78 0.66]
 [0.68 0.47 0.66 0.49 0.07 0.16 0.36 0.35 0.9  0.19]
 [0.59 0.17 0.52 0.46 0.33 0.57 0.44 0.39 0.23 0.63]]
```

Computing for Layer-1

$$X^{(1)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(0)} * W^{(0)}\right)$$

```
X_1

[[3.82 2.41 3.74 3.02 1.71 3.27 1.96 3.12 3.56 2.75]
 [3.83 2.46 3.39 2.9  1.75 3.07 1.67 3.04 3.35 2.5 ]
 [3.59 2.42 3.2  2.49 1.69 2.75 1.77 2.86 3.39 2.27]
 [4.01 2.59 3.72 3.06 1.87 3.28 1.93 3.29 3.64 2.6 ]
 [4.06 2.71 3.55 2.96 1.95 3.21 1.75 3.3  3.59 2.46]
 [4.09 2.76 3.76 3.04 1.87 3.31 1.9  3.28 3.77 2.67]
 [3.44 2.33 2.88 2.39 1.62 2.52 1.64 2.65 3.12 2.12]
 [3.21 2.12 2.88 2.19 1.55 2.46 1.74 2.58 3.05 2.05]
 [3.64 2.37 3.42 2.44 1.64 2.81 1.92 2.8  3.48 2.47]
 [3.44 2.17 3.21 2.31 1.69 2.79 1.97 2.79 3.24 2.33]]

```

Computing for Layer-2

$$X^{(2)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(1)} * W^{(0)}\right)$$

```
X_2

[[17.2  11.71 15.91 13.06  7.85 13.59  9.85 13.92 17.39 12.09]
 [17.81 12.14 16.46 13.49  8.12 14.04 10.2  14.4  18.03 12.47]
 [16.12 11.03 14.75 12.14  7.23 12.53  9.11 12.89 16.29 11.16]
 [17.67 12.06 16.4  13.42  8.   13.99 10.04 14.28 17.89 12.41]
 [18.17 12.4  16.84 13.77  8.22 14.35 10.33 14.67 18.41 12.71]
 [17.34 11.85 16.06 13.14  7.82 13.68  9.83 13.97 17.56 12.14]
 [15.26 10.44 13.95 11.47  6.81 11.84  8.55 12.16 15.38 10.54]
 [15.62 10.68 14.21 11.75  6.99 12.06  8.81 12.45 15.75 10.75]
 [15.7  10.75 14.29 11.8   7.03 12.12  8.89 12.52 15.87 10.81]
 [15.24 10.39 13.78 11.45  6.81 11.7   8.58 12.14 15.34 10.4 ]]
```

- **Iteration 2**

```
 W_1 (Weight Matrix)
====================

[[0.51 0.24 0.03 0.07 0.91 0.26 0.14 0.72 0.5  0.2 ]
 [0.42 0.55 0.07 0.75 0.08 0.81 0.71 0.29 0.6  0.06]
 [0.76 0.64 0.48 0.61 0.09 0.8  0.47 0.85 0.77 0.94]
 [0.51 0.42 0.56 0.77 0.95 0.93 0.71 0.81 1.   0.04]
 [0.12 0.09 0.85 0.29 0.15 0.62 0.18 0.44 0.37 0.7 ]
 [0.67 0.72 0.67 0.64 0.45 0.26 0.83 0.83 0.8  0.04]
 [0.9  0.32 0.71 0.02 0.76 0.29 0.24 0.76 0.46 0.08]
 [0.84 0.76 0.91 0.37 0.78 0.99 0.08 0.75 0.6  0.09]
 [0.61 0.43 0.68 0.97 0.94 0.26 0.8  0.64 0.29 0.47]
 [0.92 0.41 0.95 0.72 0.88 0.34 0.15 0.8  0.24 0.69]]
```

Computing for Layer-1

$$X^{(1)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(0)} * W^{(1)}\right)$$

```
X_1

[[86.57 64.32 76.86 73.33 84.88 74.08 60.86 95.59 77.24 45.14]
 [86.06 63.96 76.38 72.94 84.37 73.65 60.54 95.03 76.8  44.87]
 [78.06 58.01 69.17 66.22 76.63 66.85 54.99 86.2  69.69 40.67]
 [87.92 65.35 78.03 74.52 86.19 75.24 61.86 97.08 78.47 45.85]
 [87.13 64.77 77.29 73.88 85.4  74.56 61.32 96.21 77.77 45.43]
 [85.15 63.29 75.53 72.19 83.51 72.88 59.93 94.02 76.   44.39]
 [76.25 56.67 67.55 64.7  74.88 65.31 53.73 84.21 68.08 39.73]
 [75.8  56.33 67.15 64.3  74.46 64.93 53.41 83.7  67.68 39.47]
 [76.97 57.19 68.19 65.29 75.6  65.93 54.23 84.99 68.72 40.09]
 [74.87 55.64 66.33 63.51 73.6  64.16 52.76 82.7  66.87 38.97]]
```

Computing for Layer-2

$$X^{(2)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(1)} * W^{(1)}\right)$$

```
X_2

[[447.65 339.21 431.21 372.56 437.62 421.8  312.95 506.62 422.23 238.1 ]
 [448.67 339.98 432.2  373.41 438.63 422.76 313.67 507.78 423.2  238.63]
 [407.3  308.62 392.44 338.98 398.27 383.8  284.79 461.   384.22 216.58]
 [451.86 342.4  435.26 376.07 441.76 425.76 315.91 511.39 426.21 240.31]
 [450.3  341.21 433.76 374.76 440.24 424.28 314.82 509.62 424.73 239.47]
 [436.63 330.85 420.62 363.39 426.9  411.41 305.27 494.16 411.85 232.19]
 [398.26 301.77 383.75 331.46 389.45 375.29 278.49 450.77 375.7  211.77]
 [396.64 300.54 382.2  330.11 387.87 373.77 277.36 448.94 374.18 210.91]
 [399.49 302.7  384.94 332.49 390.65 376.45 279.35 452.17 376.86 212.43]
 [391.17 296.4  376.96 325.57 382.52 368.62 273.54 442.76 369.03
208.01]]
```

- **Iteration 3**

```
 W_2 (Weight Matrix)
==================

[[0.76 0.56 0.01 0.63 0.35 0.83 0.44 0.45 0.63 0.98]
 [0.78 0.83 0.38 0.03 0.16 0.68 0.2  0.49 0.9  0.56]
 [0.87 0.73 0.68 0.74 0.73 1.   0.1  0.2  0.09 0.18]
 [0.02 0.03 0.04 0.24 0.25 0.8  0.65 0.33 0.2  0.2 ]
 [0.66 0.88 0.28 0.49 0.5  0.45 0.82 0.27 0.35 0.4 ]
 [0.53 0.02 0.1  0.78 0.19 0.55 0.9  0.37 0.67 0.56]
 [0.39 0.48 0.25 0.64 0.91 0.86 0.09 0.05 0.46 0.84]
 [0.95 0.13 0.61 0.02 0.3  0.08 0.56 0.11 0.59 0.31]
 [0.45 0.39 0.94 0.45 0.79 0.96 0.28 0.53 0.68 0.04]
 [0.19 0.12 0.06 0.97 0.56 0.55 0.55 0.77 0.27 0.56]]
```

Computing for Layer-1

$$X^{(1)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(0)} * W^{(2)}\right)$$

```
X_1

[[2347.62 1661.33 1412.4  1878.96 1822.39 2604.1  1855.61 1333.28 1932.1
  1764.76]
 [2336.49 1653.46 1405.71 1870.04 1813.75 2591.75 1846.81 1326.95
1922.94  1756.39]
 [2134.89 1510.81 1284.45 1708.67 1657.27 2368.09 1687.41 1212.38
1756.95  1604.77]
 [2357.81 1668.55 1418.53 1887.1  1830.3  2615.4  1863.67 1339.05
1940.49  1772.42]
 [2336.81 1653.69 1405.9  1870.29 1814.   2592.09 1847.06 1327.11
1923.19  1756.62]
 [2281.68 1614.68 1372.74 1826.16 1771.21 2530.93 1803.47 1295.79 1877.8
  1715.16]
 [2098.63 1485.16 1262.64 1679.64 1629.13 2327.87 1658.75 1191.78 1727.1
  1577.51]
 [2087.33 1477.16 1255.85 1670.6  1620.36 2315.34 1649.81 1185.36 1717.8
  1569.01]
 [2100.78 1486.68 1263.94 1681.37 1630.8  2330.26 1660.45 1193.
1728.87  1579.12]
 [2062.92 1459.89 1241.17 1651.07 1601.42 2288.27 1630.51 1171.5
1697.71  1550.66]]
```

Computing for Layer-2

$$X^{(2)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(1)} * W^{(2)}\right)$$

```
X_2

[[10119.67  7426.02  5667.96  9714.74  8555.61 12704.    8979.29
6833.23  9252.15  9054.34]
 [10079.45  7396.5   5645.42  9676.12  8521.59 12653.49  8943.59
6806.07  9215.37  9018.34]
 [ 9256.73  6792.82  5184.59  8886.29  7825.98 11620.65  8213.55  6250.5
  8463.14  8282.21]
 [10134.81  7437.13  5676.43  9729.27  8568.4  12723.    8992.71
6843.45  9265.99  9067.88]
 [10055.79  7379.15  5632.17  9653.41  8501.59 12623.8   8922.59  6790.0
  9193.74  8997.18]
 [ 9826.73  7211.07  5503.87  9433.51  8307.92 12336.24  8719.34
6635.41  8984.31  8792.23]
 [ 9110.94  6685.84  5102.93  8746.33  7702.71 11437.63  8084.18
6152.05  8329.84  8151.77]
 [ 9067.53  6653.99  5078.61  8704.65  7666.01 11383.13  8045.66
6122.74  8290.15  8112.93]
 [ 9114.04  6688.12  5104.66  8749.3   7705.33 11441.52  8086.93
6154.15  8332.67  8154.54]
 [ 8973.33  6584.87  5025.85  8614.22  7586.37 11264.88  7962.08
6059.13  8204.02  8028.64]]
```

Run 3 (GCN, Layers = 3)

- Initialization

```
 Raw A (Adjacency Matrix)
==========================

[[0 1 0 0 0 0 0 0 0 0]
 [1 0 0 1 0 1 0 0 0 0]
 [0 0 0 0 0 1 1 1 1 0]
 [0 1 0 0 1 0 0 0 0 0]
 [0 0 0 1 0 1 0 0 0 0]
 [0 1 1 0 1 0 0 0 0 0]
 [0 0 1 0 0 0 0 1 0 0]
 [0 0 1 0 0 0 1 0 1 1]
 [0 0 1 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0 0]]

 A cap (A = A + I)
===============

[[1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 0. 1. 0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 1. 1. 1. 1. 0.]
 [0. 1. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 1. 1. 0. 1. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 1. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 1. 0. 0. 0. 0. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 1.]]

 D (Degree Matrix)
===================

[[3. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 4. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 5. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 3. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 3. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 4. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 3. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 5. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 3. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 2.]]

 X (Node Representation)
========================

[[0.5  0.49 0.2  0.75 0.14 0.51 0.65 0.09 0.25 0.82]
 [0.78 0.47 0.54 0.76 0.78 0.88 0.04 0.88 1.   0.98]
 [0.84 0.74 0.11 0.17 0.63 0.07 0.7  0.12 0.92 0.97]
 [0.56 0.1  0.03 0.47 0.85 0.66 0.95 0.3  0.48 0.87]
 [0.49 0.91 0.84 0.78 0.82 0.45 0.74 0.2  0.86 0.73]
 [0.96 0.82 0.48 0.84 0.99 0.75 0.98 0.67 0.44 0.03]
 [0.32 0.04 0.73 0.34 0.69 0.5  0.99 0.04 0.27 0.08]
```

```
 [0.74 1.   0.67 0.92 0.23 0.62 0.69 0.88 0.7  0.6 ]
 [0.4  0.53 0.83 0.84 0.46 0.18 0.26 0.95 0.96 0.07]
 [0.24 0.4  0.76 0.09 0.63 0.21 0.61 0.52 0.13 0.79]]
```

- **Iteration 1**

```
 W_0 (Weight Matrix)
==================

[[0.85 0.92 0.17 0.87 0.19 0.36 0.23 0.33 0.75 0.9 ]
 [0.42 0.36 0.82 0.14 0.17 0.77 0.   0.37 0.43 0.33]
 [0.12 0.37 0.24 0.09 0.47 0.19 0.91 0.56 0.73 0.15]
 [0.4  0.14 0.74 0.1  0.59 0.63 0.26 0.56 0.22 0.56]
 [0.65 0.51 0.87 0.74 0.36 0.93 0.09 0.88 0.52 0.29]
 [0.68 0.46 0.66 0.98 0.17 0.71 0.1  0.71 0.77 0.2 ]
 [0.95 0.52 0.1  0.28 0.47 0.09 0.19 0.43 0.5  0.11]
 [0.81 0.1  0.94 0.35 0.08 0.58 0.6  0.45 0.78 0.66]
 [0.68 0.47 0.66 0.49 0.07 0.16 0.36 0.35 0.9  0.19]
 [0.59 0.17 0.52 0.46 0.33 0.57 0.44 0.39 0.23 0.63]]
```

Computing for Layer-1

$$X^{(1)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(0)} * W^{(0)}\right)$$

```
X_1

[[3.82 2.41 3.74 3.02 1.71 3.27 1.96 3.12 3.56 2.75]
 [3.83 2.46 3.39 2.9  1.75 3.07 1.67 3.04 3.35 2.5 ]
 [3.59 2.42 3.2  2.49 1.69 2.75 1.77 2.86 3.39 2.27]
 [4.01 2.59 3.72 3.06 1.87 3.28 1.93 3.29 3.64 2.6 ]
 [4.06 2.71 3.55 2.96 1.95 3.21 1.75 3.3  3.59 2.46]
 [4.09 2.76 3.76 3.04 1.87 3.31 1.9  3.28 3.77 2.67]
 [3.44 2.33 2.88 2.39 1.62 2.52 1.64 2.65 3.12 2.12]
 [3.21 2.12 2.88 2.19 1.55 2.46 1.74 2.58 3.05 2.05]
 [3.64 2.37 3.42 2.44 1.64 2.81 1.92 2.8  3.48 2.47]
 [3.44 2.17 3.21 2.31 1.69 2.79 1.97 2.79 3.24 2.33]]
```

Computing for Layer-2

$$X^{(2)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(1)} * W^{(0)}\right)$$

```
X_2

[[17.2  11.71 15.91 13.06  7.85 13.59  9.85 13.92 17.39 12.09]
 [17.81 12.14 16.46 13.49  8.12 14.04 10.2  14.4  18.03 12.47]
 [16.12 11.03 14.75 12.14  7.23 12.53  9.11 12.89 16.29 11.16]
 [17.67 12.06 16.4  13.42  8.   13.99 10.04 14.28 17.89 12.41]
 [18.17 12.4  16.84 13.77  8.22 14.35 10.33 14.67 18.41 12.71]
 [17.34 11.85 16.06 13.14  7.82 13.68  9.83 13.97 17.56 12.14]
 [15.26 10.44 13.95 11.47  6.81 11.84  8.55 12.16 15.38 10.54]
 [15.62 10.68 14.21 11.75  6.99 12.06  8.81 12.45 15.75 10.75]
 [15.7  10.75 14.29 11.8   7.03 12.12  8.89 12.52 15.87 10.81]
 [15.24 10.39 13.78 11.45  6.81 11.7   8.58 12.14 15.34 10.4 ]]
```

Computing for Layer-3

$$X^{(3)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(2)} * W^{(0)}\right)$$

```
X_3

[[82.69 56.03 75.3  61.45 37.35 63.5  46.72 65.73 82.99 56.68]
 [82.17 55.69 74.86 61.08 37.13 63.12 46.46 65.33 82.5  56.35]
 [74.61 50.63 67.94 55.42 33.66 57.27 42.11 59.23 74.91 51.2 ]
 [83.94 56.9  76.48 62.39 37.92 64.48 47.47 66.74 84.3  57.56]
 [83.16 56.38 75.78 61.82 37.58 63.89 47.05 66.12 83.54 57.04]
 [81.32 55.14 74.08 60.43 36.73 62.45 45.97 64.63 81.67 55.77]
 [72.9  49.48 66.38 54.15 32.88 55.95 41.14 57.86 73.19 50.04]
 [72.49 49.21 66.   53.83 32.69 55.62 40.88 57.51 72.77 49.76]
 [73.61 49.96 67.01 54.65 33.2  56.47 41.51 58.4  73.88 50.51]
 [71.66 48.64 65.22 53.19 32.3  54.96 40.37 56.82 71.9  49.18]]
```

- **Iteration 2**

```
 W_1 (Weight Matrix)
====================

[[0.51 0.24 0.03 0.07 0.91 0.26 0.14 0.72 0.5  0.2 ]
 [0.42 0.55 0.07 0.75 0.08 0.81 0.71 0.29 0.6  0.06]
 [0.76 0.64 0.48 0.61 0.09 0.8  0.47 0.85 0.77 0.94]
 [0.51 0.42 0.56 0.77 0.95 0.93 0.71 0.81 1.   0.04]
 [0.12 0.09 0.85 0.29 0.15 0.62 0.18 0.44 0.37 0.7 ]
 [0.67 0.72 0.67 0.64 0.45 0.26 0.83 0.83 0.8  0.04]
 [0.9  0.32 0.71 0.02 0.76 0.29 0.24 0.76 0.46 0.08]
 [0.84 0.76 0.91 0.37 0.78 0.99 0.08 0.75 0.6  0.09]
 [0.61 0.43 0.68 0.97 0.94 0.26 0.8  0.64 0.29 0.47]
 [0.92 0.41 0.95 0.72 0.88 0.34 0.15 0.8  0.24 0.69]]
```

Computing for Layer-1

$$X^{(1)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(0)} * W^{(1)}\right)$$

```
X_1

[[399.05 296.32 353.7  337.96 391.83 341.68 280.63 440.73 356.05 208.34]
 [399.97 296.99 354.5  338.74 392.72 342.45 281.28 441.74 356.86 208.82]
 [363.26 269.74 321.91 307.73 356.68 311.02 255.5  401.18 324.1  189.69]
 [402.76 299.06 356.99 341.1  395.46 344.84 283.24 444.82 359.35 210.28]
 [401.37 298.03 355.75 339.93 394.1  343.65 282.27 443.28 358.11 209.55]
 [389.25 289.03 344.99 329.68 382.19 333.27 273.75 429.89 347.28 203.23]
 [355.24 263.78 314.8  300.95 348.8  304.15 249.87 392.32 316.94 185.51]
 [353.83 262.74 313.54 299.76 347.42 302.94 248.88 390.77 315.68 184.77]
 [356.35 264.61 315.77 301.88 349.89 305.1  250.65 393.55 317.92 186.09]
 [349.02 259.17 309.26 295.69 342.7  298.82 245.5  385.45 311.39
182.27]]
```

Computing for Layer-2

$$X^{(2)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(1)} * W^{(1)}\right)$$

```
X_2

[[2074.62 1571.96 1999.09 1726.55 2028.44 1954.73 1450.16 2348.06
1956.67 1103.64]
 [2064.72 1564.45 1989.55 1718.31 2018.77 1945.4  1443.24 2336.86
1947.34 1098.37]
 [1887.38 1430.05 1818.67 1570.71 1845.43 1778.3  1319.27 2136.14
1780.08 1003.98]
 [2083.46 1578.65 2007.6  1733.89 2037.08 1963.06 1456.33 2358.06
1965.01 1108.34]
 [2064.91 1564.59 1989.72 1718.45 2018.95 1945.58 1443.36 2337.07
1947.51 1098.46]
 [2016.48 1527.89 1943.07 1678.16 1971.62 1899.95 1409.51 2282.27
1901.84 1072.69]
 [1855.51 1405.9  1787.97 1544.19 1814.28 1748.27 1296.99 2100.07
1750.02  987.02]
 [1845.62 1398.41 1778.44 1535.96 1804.62 1738.96 1290.08 2088.89 1740.7
  981.75]
 [1857.42 1407.35 1789.81 1545.78 1816.15 1750.08 1298.33 2102.24
1751.83  988.03]
 [1824.27 1382.23 1757.87 1518.19 1783.75 1718.84 1275.16 2064.72
1720.56  970.39]]
```

Computing for Layer-3

$$X^{(3)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(2)} * W^{(1)}\right)$$

```
X_3

[[11094.92  8460.39 10655.43  9279.92 10717.31 10367.61  7816.47
12533.73  10441.05  5930.76]
 [11050.91  8426.84 10613.17  9243.11 10674.8  10326.49  7785.47
12484.02  10399.64  5907.24]
 [10153.1   7742.22  9750.97  8492.18  9807.55  9487.58  7152.97
11469.81  9554.78   5427.36]
 [11111.04  8472.69 10670.91  9293.4  10732.88 10382.68  7827.83
12551.94  10456.22  5939.38]
 [11024.76  8406.9  10588.06  9221.24 10649.54 10302.06  7767.05
12454.48  10375.03  5893.27]
 [10774.87  8216.35 10348.08  9012.23 10408.16 10068.56  7591.
12172.19  10139.88  5759.7 ]
 [ 9994.04  7620.93  9598.21  8359.14  9653.91  9338.96  7040.91
11290.13  9405.1    5342.34]
 [ 9946.85  7584.95  9552.89  8319.67  9608.32  9294.86  7007.67
11236.81  9360.69   5317.11]
 [ 9997.47  7623.55  9601.5   8362.    9657.21  9342.16  7043.32
11293.99  9408.32   5344.17]
 [ 9844.41  7506.84  9454.52  8233.99  9509.37  9199.14  6935.5  11121.1
  9264.29   5262.36]]
```

- **Iteration 3**

```
 W_2 (Weight Matrix)
====================

[[0.76 0.56 0.01 0.63 0.35 0.83 0.44 0.45 0.63 0.98]
 [0.78 0.83 0.38 0.03 0.16 0.68 0.2  0.49 0.9  0.56]
 [0.87 0.73 0.68 0.74 0.73 1.   0.1  0.2  0.09 0.18]
 [0.02 0.03 0.04 0.24 0.25 0.8  0.65 0.33 0.2  0.2 ]
 [0.66 0.88 0.28 0.49 0.5  0.45 0.82 0.27 0.35 0.4 ]
 [0.53 0.02 0.1  0.78 0.19 0.55 0.9  0.37 0.67 0.56]
 [0.39 0.48 0.25 0.64 0.91 0.86 0.09 0.05 0.46 0.84]
 [0.95 0.13 0.61 0.02 0.3  0.08 0.56 0.11 0.59 0.31]
 [0.45 0.39 0.94 0.45 0.79 0.96 0.28 0.53 0.68 0.04]
 [0.19 0.12 0.06 0.97 0.56 0.55 0.55 0.77 0.27 0.56]]
```

Computing for Layer-1

$$X^{(1)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(0)} * W^{(2)}\right)$$

```
X_1

[[57823.75 40931.65 34795.22 46290.59 44939.89 64252.09 45656.31
32885.04  47644.07 43548.44]
 [57522.54 40718.43 34613.97 46049.45 44705.79 63917.39 45418.48
32713.74  47395.88 43321.59]
 [53161.05 37631.05 31989.48 42557.91 41316.12 59071.04 41974.75
30233.32  43802.22 40036.84]
 [57806.29 40919.29 34784.72 46276.61 44926.32 64232.69 45642.53
32875.11  47629.68 43535.29]
 [57325.49 40578.95 34495.4  45891.71 44552.65 63698.44 45262.9
32601.67  47233.52 43173.19]
 [56179.49 39767.72 33805.8  44974.29 43662.   62425.04 44358.04
31949.93  46289.27 42310.11]
 [52419.38 37106.04 31543.18 41964.17 40739.7  58246.92 41389.14
29811.52  43191.11 39478.27]
 [52188.64 36942.71 31404.34 41779.46 40560.38 57990.53 41206.96 29680.3
  43001.   39304.49]
 [52425.34 37110.26 31546.77 41968.95 40744.34 58253.54 41393.85
29814.91  43196.03 39482.76]
 [51710.29 36604.09 31116.49 41396.51 40188.61 57458.99 40829.26
29408.25  42606.85 38944.23]]
```

Computing for Layer-2

$$X^{(2)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(1)} * W^{(2)}\right)$$

```
X_2

[[249350.52 182927.61 139671.08 239399.5  210779.41 312977.05 221331.44
168415.05 227985.45 223059.77]
 [248095.94 182007.23 138968.34 238194.98 209718.9  311402.34 220217.83
167567.68 226838.36 221937.47]
 [230534.84 169124.11 129131.67 221334.71 194874.26 289360.19 204630.06
155706.64 210781.93 206227.94]
 [249041.11 182700.62 139497.76 239102.43 210517.86 312588.68 221056.79
168206.07 227702.55 222782.98]
 [247103.87 181279.43 138412.64 237242.5  208880.29 310157.12 219337.24
166897.62 225931.29 221049.99]
 [242531.61 177925.14 135851.53 232852.71 205015.29 304418.15 215278.75
163809.44 221750.79 216959.81]
 [227570.27 166949.25 127471.1  218488.44 192368.27 285639.15 201998.61
153704.33 208071.37 203575.94]
 [226666.95 166286.56 126965.11 217621.17 191604.68 284505.33 201196.79
153094.21 207245.45 202767.86]
 [227578.87 166955.56 127475.92 218496.7  192375.54 285649.95 202006.24
153710.14 208079.23 203583.64]
 [224799.83 164916.81 125919.27 215828.57 190026.38 282161.78 199539.48
151833.13 205538.31 201097.61]]
```

Computing for Layer-3

$$X^{(3)} = ReLU\left(\bar{D}^{-1} * \bar{A} * X^{(2)} * W^{(2)}\right)$$

```
X_3

[[1153970.34  831312.89  653568.63 1131049.5   994744.94 1466457.64
1069569.28  801289.43 1077401.69 1047814.26]
 [1148122.62  827100.23  650256.68 1125317.93  989704.09 1459026.4
1064149.26  797228.91 1071941.98 1042504.48]
 [1072534.7   772647.18  607446.31 1051231.39  924545.83 1362969.8
994089.82  744742.46 1001369.5    973870.06]
 [1151955.87  829861.68  652427.7  1129075.04  993008.42 1463897.66
1067702.14  799890.63 1075520.88  1045985.1]
 [1143343.24  823657.19  647549.81 1120633.48  985584.17 1452952.79
1059719.44  793910.22 1067479.72 1038164.76]
 [1124031.17  809744.91  636612.12 1101705.    968936.79 1428411.15
1041819.85  780500.38 1049449.06 1020629.26]
 [1059908.37  763551.25  600295.19 1038855.85  913661.68 1346924.33
982386.97  735975.03  989580.95   962405.25]
 [1056067.27  760784.15  598119.73 1035091.05  910350.59 1342043.1
978826.81  733307.87  985994.72   958917.51]
 [1059921.68  763560.84  600302.73 1038868.9   913673.16 1346941.25
982399.31  735984.28  989593.38   962417.34]
 [1048188.39  755108.25  593657.4  1027368.66  903558.84 1332030.67
971524.19  727836.96  978638.62   951763.42]]
```

Part 2.

In terms of capturing structural information of the graph while maintaining distinct enough representations for each node, **GCN with 2-layers performs the best** for the given graph G.

- Nodes 7 & 9 are structurally congruent, with only minor differences between their initial feature representations. Trefore, it is expected that the optimal model is able to capture this similarity while maintaining the uniqueness of other node representations.
- GCN-with-3-Layers is able to generate very similar embeddings for Nodes 7 & 9, however, **due to over-smoothing**, Node-8 and a few other nodes become increasingly similar to them.
- GCN-with-1-Layer maintains the difference between other nodes, but fails to bring the representation of Nodes 7 & 9 close, as its **receptive field is highly restrictive** (only 1-hop receptors).

Part 3.

- Over-Smoothing Problem:

  Oversmoothing usually occurs when the localized graph information (obtained from lower-hop receptive fields) is overshadowed by the overall graph structure (larger-hop receptive fields). This leads to overly similar embeddings for nodes, thus losing key structural and semantic information.

  In the given graph G, it is seen that GCNs with a higher layer count (i.e. layers=3) ends up oversmoothing substantially, leading to similar embeddings for Nodes 7, 8, and 9, when Node 8 is considerably distinct from Nodes 7 & 9.

- Receptive Fields:

  The k-hop Receptive Fields for Node 1 in G have been enumerated below:

```
Node 1:
    k=1: [2]
    k=2: [2,4,6]
    k=3: [2,3,4,5,6]
    k=4: [2,3,4,5,6,7,8,9]
    k=5: [2,3,4,5,6,7,8,9,10]
```

- Skip-Connections:

  Skip connection are **used to curb over-smoothing**, by parallely making available the pervious state (with a lower-hop receptive field, i.e. local) in the embeddings generated in the next layer. In our graph G, Skip Connection could prove useful for higher layer counts of GCNs.

Q3.

Part 1.

[Please find the relevant code in `Assignment2_TiDL/gnn.py: class GNNLayer`]

Part 2.

[Please find the relevant code in `Assignment2_TiDL/gnn.py: class GCNLayer and class GCN`,

and the code to train and evaluate the above models in `Assignment2_TiDL/train_gnn.py`]

- Given below are the model performances on the three adjacency matrix normalization methods:

  [Note: All results below pertain to models with hidden dimensionality 128]

Test Accuracy (%)	Row Norm	Column Norm	Symmentric Norm
GCN (1-Layer)	74.44	73.33	73.37
GCN (2-Layers)	72.61	72.44	72.37
GCN (3-Layers)	72.65	72.56	72.44

  Although the absolute difference between the test accuracies is not substantial (possibly due to the low hidden dimensionality), it is possible to comment on the effect of the various normalization methods:

  - Row Normalization: This is effectively taking the mean value of one's neighbours. This ensures easier optimization (marginally visible in the above results as well), as it helps bring all node values in a similar range despte the depth of the network. The clear disadvantage is however, that this can lose out on key structural aspects of individual nodes.

$$A_{row} = D^{-1}A$$

  - Column Normalization: This is similar to the PageRank algorithm, where we effectively compute the probability of starting at a node and ending up on the current node. Although this helps capture more granular node level information, the relative ranges of different node values can vary heavily.

$$A_{col} = AD^{-1}$$

  - Symmetric Normalization: Symmetric Normalization is aimed at capturing the best of both worlds (row & column), however, in our specific case (G), this doesn't seem to help much. This could again be due to the low hidden dimensionality. A more in-depth analysis is needed.

$$A_{row} = D^{-1/2}AD^{-1/2}$$

Part 3.

For this question, we have implemented the **GraphSage** model.

[Please find the relevant code in `Assignment2_TiDL/gnn.py: class GSageLayer and class GSage`,

and the code to train and evaluate the above models in `Assignment2_TiDL/train_gnn.py`]

- Given below is a comparitive study between the various configurations of the GCN and GraphSage GNN variants:

Layer Count	Hidden Dimensionality	GCN Test Accuracy (%)	GSage Test Accuracy (%)
1	128	74.74	70.78
1	256	74.21	70.84
1	512	74.55	71.08
2	128	73.33	69.78
2	256	73.33	69.80
2	512	73.36	69.70
3	128	73.40	69.80
3	256	73.38	69.80
3	512	73.36	69.80

- Observations:
  - GCN is able to perform natably better than GSage for the given grapg G (citeseer).
  - Both models show a miniscule variation in the overall scores w.r.t to the hidden dimensionality, but this could also be due to the fact that relative to the node-count of the graph (>3k), these values (i.e. 128, 256, 512) are relatively closeby.
  - Also, as we observe along the rows instead of the columns, we see that beyond a single layer of GNN, both models show a drop in performance. This could possibly be due to over-smoothing as neither models consist of skip-connections.

Part 4.

[Please find the relevant code for the RNN model in `Assignment2_TiDL/train_rnn.py: class RNNClassifier`,

and the code to train and evaluate the above model in `Assignment2_TiDL/train_rnn.py`]

- Data Cleaning:
  - We make use of the Imdb Sentiment Classification dataset
  - The data is cleaning for foreign characters (including HTML tags)
  - The text is then tokenized using the SentencePiece tokenizer, with a target vocabulary of 5000.
  - Each text is restricted to (or padded to) 64 tokens.
- Model Params:
  - The Classifier uses a single RNN
    - Single Layer
    - Unidirectional

        - With an input dimensionality of 64
        - And a hidden dimensionality of 128
    - A model-wide dropout of 0.3 is applied
- Results:
    - The model is able to converge well, with:
        - Training Accuracy: **81.14%**
        - Testing Accuracy: **74.87%** (epoch 55)

Part 5.

[Please find the relevant content in README.md]