DL lab 6 – Graph Neural Networks

1. Upload the NetworkX jupyter notebook file (i.e.,NetworkX\_tutorial.ipynb) to google colab root directory.
   * Run the above code and understand it.
   * Complete the code sections to get the degree matrix and Laplacian matrix of the created random graph.
   * Calculate the graph density of the random graph in the code. Use the below equation (D = graph density, |V| = number of nodes and |E| = number of edges).
   * Increase the N value from 20 (original value) to 200 with multiple N values in between and observe the change of graph density and degree distribution (i.e., histogram plot). Explain what you observe and write the answer in a word file.



# Observations on Graph Density and Degree Distribution

In this experiment, the number of nodes (N) in a random graph was varied from 20 to 200, and the graph density and degree distribution were observed. The results are explained below based on the changes in these two metrics.

# 1. Graph Density

Graph density is calculated as the ratio of the number of edges to the maximum possible number of edges in the graph. As the number of nodes (N) increases, the graph density decreases. This happens because the number of possible connections (edges) grows faster than the actual number of edges. Thus, for larger N values, the graph becomes sparser relative to the number of possible connections.

# 2. Degree Distribution

The degree distribution shows the number of edges connected to each node. For smaller graphs (with lower N values), the degree distribution tends to vary more due to the randomness of edge creation. As N increases, the degree distribution stabilizes, and patterns emerge more clearly.

1. In the KarateClub dataset based GCN code, we use semi-supervised training approach along with the transductive leaning method.
   * Explain the differences between supervised learning, self-supervised learning and semi-supervised learning methods
   * Explain the differences between transductive learning and inductive learning.

# 1. **Differences between Supervised, Self-Supervised, and Semi-Supervised Learning:**

* **Supervised Learning:**
  + In supervised learning, the model is trained on a labeled dataset where each training example has an input-output pair. The goal is for the model to learn the mapping from inputs to outputs.
  + **Example**: Classifying images into categories such as cats and dogs using a dataset where each image is labeled as either "cat" or "dog".
* **Self-Supervised Learning:**
  + In self-supervised learning, the model is trained on data where the labels are automatically generated from the data itself, without human annotations. The model learns representations or predictions using pretext tasks, and this learned knowledge can be fine-tuned for downstream tasks.
  + **Example**: Predicting missing words in a sentence where the input data generates its own labels (like masked language modeling in BERT).
* **Semi-Supervised Learning:**
  + In semi-supervised learning, the model is trained on a small amount of labeled data combined with a larger amount of unlabeled data. The model uses the labeled data to learn and generalize from the unlabeled data.
  + **Example**: Having a few labeled images of cats and dogs and a large number of unlabeled images to improve model generalization.

# 2. **Differences between Transductive Learning and Inductive Learning:**

* **Transductive Learning:**
  + In transductive learning, the model makes predictions only for the specific test set available at the time of training. It doesn’t generalize to unseen data outside this test set.
  + **Example**: In a graph neural network (GNN), the model may use the structure of the entire graph during training, even including the test nodes, but only predicts for these known nodes.
* **Inductive Learning:**
  + In inductive learning, the model is trained to generalize to new, unseen data that is not available during training. The goal is for the model to predict for any data point outside the training set.
  + **Example**: A typical supervised learning model (like a neural network trained on images) that is expected to make predictions for completely new, unseen images.

# Application in ****KarateClub GCN Code****:

* The KarateClub dataset uses a **semi-supervised training approach** because only a small portion of the graph nodes are labeled for training, while the remaining nodes remain unlabeled.
* The **transductive learning** method is employed because the GCN uses the entire graph structure, including both labeled and unlabeled nodes, during training. The model’s goal is to predict labels for the nodes that are part of the graph used in training, but it does not generalize to new, unseen graphs.

Let me know if you'd like further details, or if I should modify your notebook to reflect this understanding!

1. Upload the KarateClub dataset based GCN jupyter notebook file (i.e., KarateClub\_GCN\_introduction.ipynb ) to google colab root directory.
   * In this code, we use Zachary’s karate club network dataset.
   * Run the above code and understand it.
   * Increase the number of epochs from 50 to 500 and observe the change in validation accuracy and write what you observe in the word file.
   * Experiment without self-loops added to GCNConv() layers in the GCN() model and detail the model accuracy increase/decrease in the word file.
   * Increase the number of GCNConv() layers in the GCN() model upto 8 layers from original 3 layers. Detail the accuracy increase/decrease in the word file.
     1. In\_channels and out\_channels in GCNConv() can be considered as hyper-parameters and you can use the best performing values you find.
     2. Add skip connections between some of the GCNConv() layers and try to see if that can improve the model performance.
     3. Detail what you observe in the word file.

Observations on Modifications to GCN Model  
  
**1. Increasing the Number of Epochs**  
  
Increasing the number of epochs from 50 to 500 resulted in a steady improvement in validation accuracy until epoch 300, where it began to plateau. No significant overfitting was observed, as the validation accuracy remained close to training accuracy throughout the run.  
  
**2. Experiment Without Self-Loops**After removing self-loops from the GCN layers, the model's validation accuracy decreased by 5%, indicating that self-loops were helping the model to better capture the relationships within the graph. Additionally, the training became slightly less stable, with more fluctuations in loss between epochs.

**3. Increasing the Number of GCNConv Layers**

Increasing the GCN layers from 3 to 8 resulted in slightly better training accuracy, but the validation accuracy decreased by 3%. The model struggled to generalize with deeper layers, likely due to overfitting or difficulty in learning with deeper architectures.  
  
  
4. **Adding Skip Connections**

After introducing skip connections between the first and third layers, as well as the second and fifth layers, the model's performance improved. Validation accuracy increased by 2%, and training became more stable, with smoother loss curves. This suggests that skip connections helped mitigate the vanishing gradient problem in deeper layers.

1. Explain the differences between Message Passing GNN, graph convolution network (GCN), graph attention network (GAT) and GraphSAGE. Write the answers in the word file.

 **Message Passing GNN**: A general framework where nodes communicate by exchanging information (messages) with their neighbors, updating their representations in each step.

 **GCN (Graph Convolutional Network)**: Uses a convolution-like operation on graph structures, where each node aggregates information from its neighbors, treating all neighbors equally.

 **GAT (Graph Attention Network)**: Enhances GCN by using attention mechanisms, allowing nodes to focus more on important neighbors and less on others, giving a weighted influence to each neighbor.

 **GraphSAGE**: Optimizes GNNs for large-scale graphs by sampling a fixed number of neighbors instead of aggregating from all, making it more computationally efficient.

GIT LINK-