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.



**When increasing the number of nodes 𝑁 N from 20 to 200 while keeping the number of edges 𝐸 = 2 𝑁 E=2N, the graph's density significantly decreases. N, while the number of actual edges only grows linearly. For 𝑁 = 200 N=200 and 𝐸 = 400 E=400, the density is approximately 0.0201, indicating a much sparser graph compared to when 𝑁 = 20 N=20. Additionally, the degree distribution becomes more concentrated around the expected degree, which is 2 𝐸 𝑁 ≈ 4 N 2E​≈4. In larger graphs, most nodes will have degrees close to this average, with fewer nodes deviating significantly. As a result, the degree histogram will show a peak near 4, reflecting a more uniform distribution of node degrees in the larger graph.**

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. Supervised Learning:**

**- Uses fully labeled data (input-output pairs).**

**- Goal: Predict labels for new data.**

**Example: Image classification with known labels.**

**2. Self-Supervised Learning:**

**- Model generates its own labels from data.**

**- Goal: Learn useful representations.**

**Example: Predicting missing words in a sentence.**

**3. Semi-Supervised Learning:**

**- Uses a small amount of labeled data + large unlabeled data.**

**- Goal: Improve performance by using both.**

**Example: Training on few labeled images + many unlabeled.**

**4. Transductive Learning:**

**- Trains on labeled + unlabeled data and makes predictions only on the provided data (no generalization).**

**Example: GCN for node classification in a graph.**

**5. Inductive Learning:**

**- Trains on labeled data and generalizes to unseen data.**

**Example: Neural networks for classifying new images.**

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.

**Loss**: As the number of epochs increases from 50 to 500, the loss gradually decreases, indicating the model’s ability to reduce error over time. By epoch 500, the loss drops to 0.0176, suggesting significant learning and optimization. However, this alone doesn't ensure better generalization, as further training primarily refines the model’s ability to fit the training data.

**Training Accuracy**: Starting at 25% during the initial epoch, the training accuracy reaches 100% by epoch 50 and stays constant. This shows the model has effectively memorized the training data early on, which might be a sign of overfitting, as achieving perfect accuracy on the training set does not guarantee the same performance on new data.

**Validation Accuracy**: Initially at 35.29%, validation accuracy gradually improves, reaching 73.53% at epoch 60, and stabilizing around 79.41% by epoch 100. After epoch 300, it increases slightly to 82.35% and remains unchanged. This indicates that while training for more epochs improves performance, the benefits diminish after 300 epochs, suggesting that further training has little impact on validation accuracy.

**Overfitting**: The early achievement of 100% training accuracy, combined with the slower improvement in validation accuracy, indicates overfitting. While the model performs perfectly on the training data after epoch 50, the validation accuracy lags, reflecting its difficulty in generalizing to unseen data, which is a common issue when models are trained for too long.

**Optimal Epochs**: After 300 epochs, validation accuracy sees minimal improvement, suggesting diminishing returns from further training. The slight increase between epochs 300 and 500 does not justify the computational cost, making 300 epochs potentially optimal for balancing training time and performance.

* + Experiment without self-loops added to GCNConv() layers in the GCN() model and detail the model accuracy increase/decrease in the word file.

**The results showed a decrease in validation accuracy. After 500 epochs, the model without self-loops achieved a training accuracy of 100% and a validation accuracy of 61.76%. In comparison, the original model with self-loops demonstrated higher validation accuracy**

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.

This increase validation accuracy and training accuracy keeps 100%

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 GNNs allow nodes to update their features by exchanging information with neighbors iteratively. Graph Convolution Networks (GCNs) aggregate features from neighbors using a fixed weight matrix and averaging, focusing on local graph structure. Graph Attention Networks (GATs) enhance this by applying attention mechanisms to weigh the importance of neighbors' features dynamically. GraphSAGE improves scalability by sampling a fixed-size neighborhood and using various aggregation functions (mean, LSTM, pooling), making it suitable for large, inductive learning tasks. Each model has unique methods for feature aggregation and updating, affecting their performance and scalability.**

**Submission.**

Download the final modified notebook files (all 2 jupyter notebooks). Add these notebooks and the word file to a new zip file. Upload this zip file to the courseweb submission link. The file name should be your registration number.