GitHub repository-<https://github.com/Isuru-rangana/DL_Lab/tree/main/Lab06>

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.



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

 Supervised Learning: Uses a dataset where both input features and target labels are provided. The model learns to map inputs to outputs directly.

 Self-supervised Learning: Creates its own supervisory signal from the input data. It might predict missing parts of the input or solve pretext tasks related to the data structure.

 Semi-supervised Learning: Utilizes a small amount of labeled data along with a larger amount of unlabeled data. It aims to leverage the structure in the unlabeled data to improve performance over purely supervised learning on the small labeled dataset.

* + Explain the differences between transductive learning and inductive learning.
  + Transductive Learning: Makes predictions only for a specific set of unlabeled examples that are available during training. It doesn't generalize to completely new, unseen data.
  + Inductive Learning: Learns a general model that can make predictions on any new, unseen examples. It aims to extract patterns that apply broadly beyond the training set

.

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.

* Increasing epochs from 50 to 500: This would likely lead to more stable convergence, but could also result in overfitting if the model is complex enough. observe the validation accuracy improving initially, then plateauing or even decreasing slightly due to overfitting.
* Removing self-loops: This could potentially decrease accuracy, as self-loops allow nodes to retain their own features during message passing. Without them, a node's updated representation might lose important information about its original features.
* Increasing GCNConv layers to 8: This would dramatically increase the model's capacity, potentially leading to better performance if there's enough data and regularization. However, it could also cause issues like oversmoothing, where node features become too similar after many rounds of message passing.

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 update their representations by aggregating messages from neighbors. Other variants can be seen as specific implementations of this general idea.
* **GCN**: Applies a specific type of graph convolution, using a normalized adjacency matrix to weight neighbor contributions. It's efficient but can struggle with heterogeneous node degrees.
* **GAT**: Introduces attention mechanisms to dynamically weight neighbor contributions. This allows the model to focus on the most relevant neighbors for each node, potentially capturing more complex relationships.
* **GraphSAGE**: Uses sampling techniques to efficiently handle large graphs. It defines different aggregator functions to combine neighbor information, making it flexible for various graph structures.

**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.