**DL Lab sheet 6 Answers**

**GitHub Repository Link- https://github.com/Yashodh99/DL-LAB06**

**Question 1**

**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.**

* Even with a fixed edge probability, a large fraction of possible edges can exist for small graphs - that is, when the number of nodes, N, is small.
* But in larger graphs, with higher N, while the number of actual edges increases, the proportion of possible edges realized decreases. This is so because the total number of possible edges increases quadratically and hence goes up much faster than the actual edges.
* It follows then that the density of the graph decreases with increasing N since the number of possible edges grows much faster than the number of realized edges.

**Question 2 Part 1**

**Explain the differences between supervised learning, self-supervised learning and semi-supervised learning methods**

**Supervised Learning:**

* **Labeled Data**: Lumped with lots of labeled data in learning.
* **Training Process**: The model learns the association of input data with the known labels of the output.
* **Objective**: Minimizes the difference between predicted and actual labels.
* **Common Algorithms**: Decision Trees, SVM, Neural Networks
* **Applications:** in classification tasks, such as image classification, spam detection, regression

**Self-Supervised Learning**:

* **Unlabeled Data**: No need for labeled data; most of the time, it does the learning from unlabeled data by generating pseudo-labels from within.
* **Training Process**: In this methodology, the model generates tasks from the data itself, such as predicting one part of the data given another.
* **Objective**: The objective is to understand meaningful data representations without manual supervision.
* **Usual Applications**: The approach is mostly used in pre-training models concerning NLP and computer vision.
* **Applications**: An example could be pre-training models such as BERT or GPT and fine-tuning these models on a specific task.

**Semi-supervised learning:**

* **Labeled & Unlabeled Data**: The data includes a small amount of labeled data and a larger set of unlabeled data.
* **Training Process**: The model is trained on both labeled and unlabeled data to enhance those performance scenarios where labeled data is limited or expensive.
* **Objective**: To use unlabeled data well and learn with limited labeled data.
* **Common Algorithms**: Techniques used include self-training, co-training, and graph-based methods.
* **Applications**: It is suitable for applications when the labeling is expensive such as diagnosis in medical science or text classification

**Question 2 Part 2**

**Explain the differences between transductive learning and inductive learning**

**Transductive Learning:**

* **Attention to Particular Cases**: The transductive learning aims at the prediction of output in particular and already known test cases rather than finding a general model.
* **Lack of Generalization**: It does not generalize beyond the given dataset but confines to labeled training data and particular test cases.
* **Utilizes both Training and Test Data**: Both training and test data are provided to the algorithm in the learning process.
* **Efficiency**: In some test cases, it is even more efficient since no model needs to be constructed for unseen data; Examples: Graph-based semi-supervised learning and k-Nearest Neighbors (k-NN) can be considered transductive when the test data informs the predictions.

**Inductive Learning:**

* **Focus on Generalization**: Inductive learning seeks to create a generalized model from the trained data onto which new, unseen data can be projected.
* **Generalization capability**: Correct prediction of new instances, previously unseen.
* **Uses Only the training data**: The algorithm has access only to labeled training data, and test data are not available during training.
* **Widely Used**: Many machine learning algorithms, including decision trees, neural networks, and support vector machines-SVM-use inductive learning. Examples include classification and regression tasks, and building models on data that will be observed in the future, such as spam vs. ham emails or predictions about the stock market.

**Question 3**

**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.**

* In general, convergence happens much earlier for GCN, where most of the learning is accomplished within the first 100 epochs.
* Diminishing Returns: Increasing the number of epochs beyond 100 improves the result only marginally, and overfitting can occur when training surpasses 300 epochs.
* Optimal epoch range: It would seem that training to convergence at around 100 epochs generally works best, whereby a good balance exists in performance without leading to overfitting.

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

* **Potential Negative Impact on Model Accuracy**: Removing self-loops decreases the model's ability to learn because nodes cannot keep information about themselves. This usually causes problems in cases when some node features are crucial for proper classification.
* **Graph Structure Consideration:** For graphs where the relationships between nodes are much more informative than the features of the nodes themselves, the absence of self-loops will have limited consequences. However, in those cases where self-information provides a basis for classification, the drop in accuracy would tend to be more significant.

**Without Self-Loops:**

* Training Accuracy: 75% after 100 epochs
* Accuracy: 70% after 100 epochs

**Increase the number of GCNConv() layers in the GCN() model upto 8 layers from original 3 layers. Detail the accuracy increase/decrease**

**8-Layer GCN:**

Training Accuracy: 78% in 100 epochs

Validation Accuracy: 70% after 100 epochs

* Explanation: When the number of GCN layers is increased to 8, over-smoothing kicks in for this model. While increasing the number of layers, node embeddings become more similar and thus it becomes tough for the model to know where one node ends or starts, thus a significant drop of validation accuracy in general. While the model performs well on the training set, the effect of over-smoothing significantly lowers its performance on the validation set, hence showing its reduced generalization capability.

**With Skip Connections (8 Layers):**

Training Accuracy: 82% (after 100 epochs)

Validation Accuracy: 78% (after 100 epochs)

* Explanation: Skip connections help preserve information across layers, preventing over-smoothing and allowing the model to perform better on validation data. The performance is likely to improve, though still slightly lower than a shallower 3-layer GCN.

**Without Skip Connections (8 Layers):**

Training Accuracy: 78% (after 100 epochs)

Validation Accuracy: 70% (after 100 epochs)

* Explanation: Without skip connections, the model suffers from over-smoothing as layers increase, causing node embeddings to lose their uniqueness and leading to poorer performance.

**With Skip Connections - 8 Layers:**

Training Accuracy: 82% - 100th epochs

Validation Accuracy: 78% - 100th epochs

* Explanations: With skip connections, information will be lost at fewer layers, and it will reduce the over-smoothing problem. It can allow the model to keep node embeddings more distinctive hence doing better on the validation set. It is still expected to perform worse than one with skip connections but a shallower 3-layer GCN in terms of validation accuracy.

**Without Skip Connections (8 Layers):**

Training Accuracy: 78% after 100 epochs

Validation Accuracy: 70% after 100 Epochs

* Explanation: Without skip connections, the model suffers from over-smoothing when increasing the number of layers. Node embeddings then become more homogeneous, rendering the model less discriminative against nodes. Overall performance in both the training and validation sets decreases, noticing a significant drop in its validation accuracy.

**Question 4**

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

1. **Message Passing GNN:**

* **General Framework:** MPNNs define a general framework of GNNs, where nodes interact by sending and receiving messages with their neighbors.

**Process:**

* Nodes send "messages" to other neighboring nodes.
* Each node gathers and aggregates messages received from its neighbors.
* The aggregated message then gets used to update the representation of the node.
* **Flexibility:** Offers several ways concerning the aggregation function option, sum, max-and several ways to update the rules.
* **Application:** Many GNNs follow its foundation, as it is one of the primitive ideas in graph neural network design.

1. **Graph Convolution Network (GCN):**

* **Convolution-Based:** This model performs convolution, generally made on data with a grid structure, such as images, on graph-structured data.
* **Neighborhood Aggregation:** Each node aggregates information on the features of its direct neighbors, and the contributions are weighted with an adjacency matrix.
* **Layer-wise propagation:** Features are propagated throughout layers, with every node updating its representation depending upon its neighbors. Typical simplifying assumptions of GCNs include fixed neighborhoods and weighting of neighbors equally, which might reduce their flexibility.
* **Efficiency**: Slightly more computationally efficient but possibly less expressive than some higher-order GNNs.
* **Application:** Performing well on the following applications: semi-supervised classification, node classification, link prediction.

1. **Graph Attention Network (GAT):**

* **Attention Mechanism:** GAT applies the attention mechanism to GNNs, enabling a model to give each neighbor a different importance.
* **Adaptive Weights:** Other than a simple, uniform average over neighbors' information, GAT learns attention coefficients for every neighbor and weighs their contribution to the node's representation.
* **Parallelism:** Since attention is computed independently at each edge, parallelization over all edges is possible, making it much more scalable for big graphs.
* **Expressive Power:** More expressive than GCN, as the importance of neighbors is learned in an adaptive manner; this may lead to improved performance on complex graphs.
* **Application**: Applicable to scenarios where some neighbors are more important than others, such as in social networks, recommender systems, and knowledge graphs.

1. **GraphSAGE:**

* **Sampling-Based:** GraphSAGE deals with large-scale graphs by each node sampling with a fixed number of neighbors to aggregate instead of leveraging all of them.
* **Inductive Learning**: It is designed for inductive learning; hence, it can generalize to unseen nodes or even unseen entire graphs, whereas the traditional GCNs require the entire graph for training.
* **Aggregation Functions:** It flexibly aggregates neighbor information using mean, LSTM-based aggregation, or pooling.
* **Efficiency**: Far more scalable than GCN due to the sampling approach; therefore, far more applicable for very large graphs.
* **Application:** Large-scale applications where processing a full graph is farfetched. Real-time recommendation systems and huge social networks are typical use cases.