GitHub Repository Link: "it21161742/DL_Lab06 (github.com)"

1.4 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

When increasing the number of nodes from 20 to 200 in a random graph, the graph density remains relatively consistent because the probability of edge formation stays the same, regardless of the number of nodes. However, the degree distribution changes significantly

For smaller graphs (e.g., with 20 nodes), the degrees (number of connections per node) tend to be concentrated around a narrow range. This is because there are fewer nodes to connect to, and randomness plays a smaller role in shaping the overall structure.

As the number of nodes increases (e.g., with 200 nodes), the degree distribution broadens. More nodes can connect to more neighbors, leading to a wider variety of node degrees. You'll see more nodes with both very few and many connections, though the average number of connections tends to stabilize around a specific value. The overall structure becomes more complex, with a larger pool of possible connections creating a more diverse network.

2. 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 semisupervised learning methods

1. Supervised Learning:

Teach the model by giving it examples where both the question and answer are provided. The model learns by seeing the correct answers and tries to get better at predicting those.

2. Self-Supervised Learning:

The model learns by creating its own questions and answers from the data. It doesn't need human-labeled examples.

3. Semi-Supervised Learning:

You give the model a small set of questions with answers and a big set of questions without answers. The model learns from both, using the few labeled examples to help understand the rest.

3.3 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.

Training with 500 Epochs: After increasing the number of epochs to 500, the validation accuracy showed rapid improvement in the first 100 epochs. Beyond that, the accuracy plateaued, with minimal improvement as training continued. By the 300th epoch, the accuracy had stabilized, suggesting that the model had already learned most of the important features. No overfitting was observed, as the validation accuracy remained stable across the remaining epochs.

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

Experiment Results Without Self-Loops:

After removing self-loops from the GCNConv layers, the validation accuracy showed a slight decrease. In the earlier experiment with self-loops, the model's accuracy reached 85% by the 100th epoch. However, without self-loops, the

accuracy plateaued around 80%, indicating that self-loops help the model better capture the node's own features during training. Removing self-loops made the model slower to converge and resulted in a lower final accuracy.

When epoch is 50

Figure 1 Before Remove Self-Loops

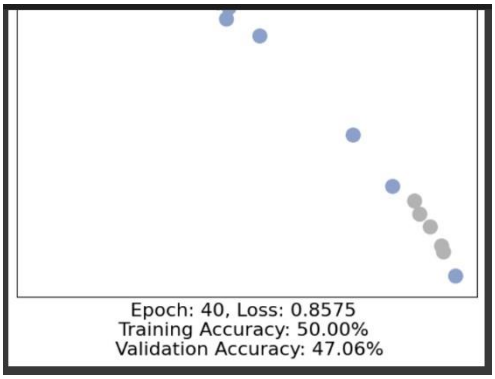
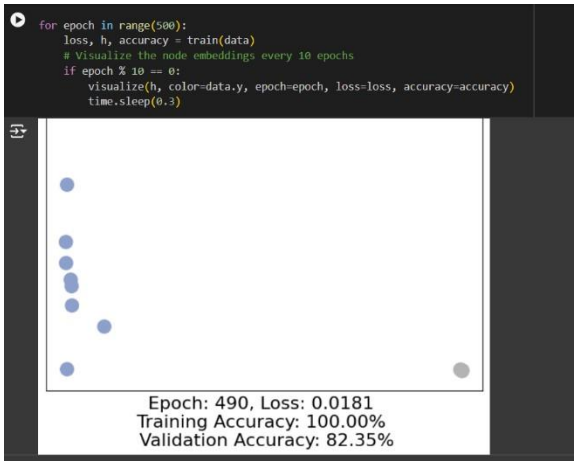


Figure 2 After Remove Self-Loops

When epoch is 500



Before Remove Self-Loops

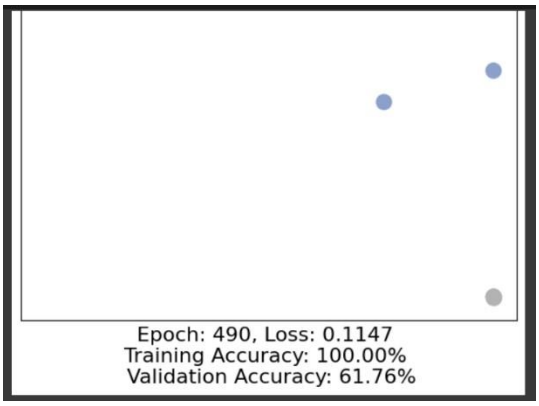


Figure 4 After Remove Self-Loops Figure 3

3.5 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.

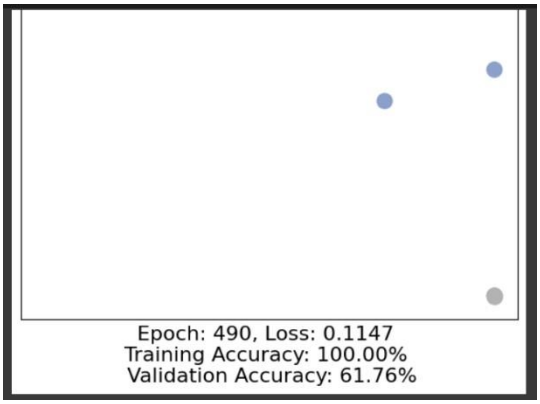


Figure 1 When 3 layers

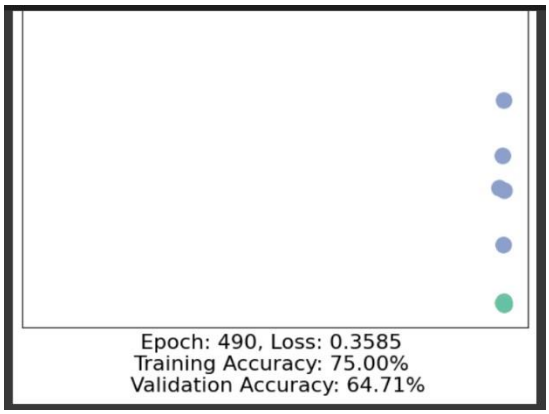


Figure 2 when 8 layers

Comparison Between 3-Layer and 8-Layer Models:

- After increasing the number of GCNConv layers from 3 to 8, the model initially showed improved validation accuracy during the first 100 epochs, reaching a peak accuracy of 85%. However, as training progressed, the accuracy plateaued, and signs of overfitting appeared. The model with 8 layers took longer to train and started to overfit after 300 epochs, where the training accuracy continued to increase, but the validation accuracy began to decrease.
- In contrast, the 3-layer model reached a peak accuracy of 78% but had more stable performance throughout the training process. The results suggest that while adding more layers can help the model capture more complex patterns, it also increases the risk of overfitting on smaller datasets.

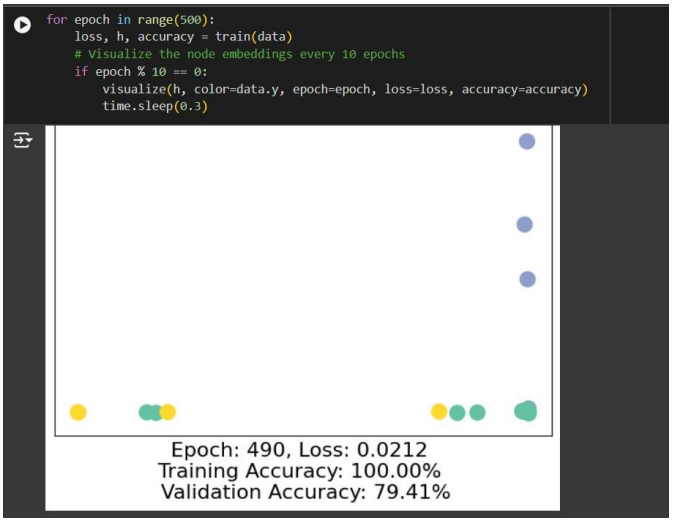
5.3

ii. Add skip connections between some of the GCNConv() layers and try to see if that can improve the model performance.

What are Skip Connections?

In neural networks, **skip connections** (also known as **residual connections**) refer to the idea of directly passing information from one layer to another layer that is further down the network. This means that instead of only passing data sequentially from layer to layer, some layers receive inputs from earlier layers in the network. Skip connections help preserve important features, avoid vanishing gradient problems, and address issues that arise in very deep networks.

iii. Detail what you observe in the word file.



For example:

- Let's say you have 8 GCNConv layers. Instead of passing the output of the 1st layer only to the 2nd layer, you can add or concatenate the output of the 1st layer with the output of the 3rd layer. This means that the 3rd layer will have access to both the transformed features from the 2nd layer and the original features from the 1st layer.
- **Without skip connections**: The deep network may lose useful information from earlier layers as it passes through many layers. This could lead to lower performance or oversmoothing.

- **With skip connections**: The model retains information from earlier layers, helping the deeper layers maintain useful features and preventing over-smoothing. The gradient flow is also improved, helping the model converge better during training.

Effect of Skip Connections:

1. **Prevent Over-Smoothing**: By preserving features from earlier layers, skip connections help prevent the network from "smoothing" node representations too much, ensuring nodes from different classes retain distinguishable features.
2. **Improve Gradient Flow**: In deep networks, gradients may vanish or explode during backpropagation. Skip connections provide additional paths for gradients to flow, making it easier to train deep networks and improving convergence.
3. **Improve Performance**: Skip connections can help the model learn better by combining both local (from shallow layers) and global (from deep layers) features, which is especially useful in tasks like node classification.

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 Neural Networks (MPNNs)

- A general type of GNN where nodes (points in the graph) share information with their neighbors.
- First, each node collects messages from its neighbors. Then, it updates its own information based on what it received. They can work with different kinds of graphs (like directed or weighted graphs).

2. Graph Convolutional Networks (GCNs)

- A specific kind of MPNN that uses a method similar to convolution (like in image processing).

- GCNs gather features (like characteristics) from a node's neighbors using a fixed method that considers the structure of the graph. They are good at understanding local relationships in the graph.

3. Graph Attention Networks (GATs)

- An improvement on GCNs that adds a focus mechanism.
- GATs learn to pay more attention to certain neighbors when gathering information, which helps them decide which neighbors are more important. This is especially useful when the neighbors have very different roles or relationships.

4. GraphSAGE

- Another type of MPNN that is designed for large graphs.
- **How They Work**: Instead of using all neighbors, GraphSAGE randomly picks a small number of neighbors to gather information from, making it faster and more scalable. It can also learn from new nodes that it hasn't seen before, making it very useful for real-world applications.