Part 2: Graph Convolutional Networks (GCN)

1. Introduction

Graph Convolutional Networks (GCNs) are a class of neural networks designed to operate on graph-structured data.

2. Implementation Approach

Architecture:

- I implemented the GCN model which consists of two GraphConvolution layers followed by ReLU activation functions.
- The first layer takes input node features and produces hidden representations, while the second layer produces the final output logits for classification.
- Hyperparameters such as input dimension, hidden dimension=16, output dimension=7, and dropout probability=0.4 are chosen based on empirical observations and common practices in GCN architectures.

Data Preparation:

- The dataset is loaded from provided files(cora.content,cora_train.cites,cora_test.cites), containing node features, adjacency information, and labels.
- Node features are represented as attribute vectors, while the graph structure is represented using adjacency matrices.
- Data preprocessing involves converting node features and labels into appropriate formats (tensors) for input to the GCN model.
- Additionally, the adjacency matrix is normalized using the below normalization method described in Kipf's paper to enhance model stability and convergence.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) .$$

3. Training Process

Procedure:

- The training process utilizes the Adam optimizer with a learning rate=0.01 for gradient descent optimization.
- Cross-entropy loss is used as the loss function to measure the difference between predicted and actual class labels.
- Dropout regularization is applied during training to prevent overfitting.
- The model is trained using a fixed number of epochs, with model parameters updated iteratively using backpropagation.
- Observations from training are periodically monitored, and adjustments to hyperparameters or training strategies may be made based on performance metrics or convergence behaviour.

4. Evaluation Results

Metrics:

- Evaluation metrics include accuracy, precision, recall, F1-score, confusion matrix, and classification report.

Accuracy: 0.95903988183161

Precision: 0.95897284364698075

Recall: 0.9545615342411945

Macro F1-score: 0.958863171627767

Confusion Matrix:

[[279 6 6 4 1 0 2]

[2 402 9 2 0 2 1]

[2 11 793 4 2 2 4]

[1 6 21 393 0 2 3]

[1 3 7 2 201 0 3]

[0 2 3 1 1 172 1]

[3 7 4 3 2 2 330]]

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.94	0.95	298
1	0.92	0.96	0.94	418
2	0.94	0.97	0.95	818
3	0.96	0.92	0.94	426
4	0.97	0.93	0.95	217
5	0.96	0.96	0.96	180
6	0.96	0.94	0.95	351
accuracy			0.96	2708
macro avg	0.96	0.95	0.95	2708
weighted avg	0.96	0.95	0.95	2708

Above are my evaluation results. When I increase the number of epochs from 7,000 and above my accuracy also improves correspondingly.

For 10,000 epochs iam getting below results:

Accuracy: 0.96003988183161

Precision: 0.9558984484256027

Recall: 0.9429591083437802

Macro F1-score: 0.9490704700565743

Confusion Matrix:

[[275 7 8 3 0 0 5]

[3 397 12 2 1 1 2]

[1 9 797 6 1 1 3]

[1 5 20 394 1 1 4]

[2 3 6 2 200 0 4]

[0 1 4 1 1 171 2]

[0 6 4 2 1 2 336]]

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.92	0.95	298
1	0.93	0.95	0.94	418
2	0.94	0.97	0.96	818
3	0.96	0.92	0.94	426
4	0.98	0.92	0.95	217
5	0.97	0.95	0.96	180
6	0.94	0.96	0.95	351
accuracy			0.96	2708
macro avg	0.96	0.94	0.95	2708
weighted avg	0.95	0.95	0.95	2708

Analysis:

- The evaluation results are analyzed to understand the model's strengths and weaknesses.
- Strengths such as high accuracy or robustness to noise, as well as weaknesses such as class imbalance or overfitting, are identified.