
# COMP8221 Assignment 2

## Project Option 1: Real-world applications of GNN

**47943319 - NIYATI**

## SECTION 1

**Why Choose the Cora Dataset?**

1. **Graph-Structured Data:** Cora is an ideal fit for GNNs because it consists of graph-structured data where nodes represent research papers and edges represent citations. GNNs excel in leveraging these graph structures to improve learning.
2. **Node Classification Task:** The task is to predict the topic (class) of each paper based on its citations and content. GNNs are known for their ability to aggregate information from neighbouring nodes to enhance classification performance.
3. **Simple and Well-Studied Benchmark:** Cora has fewer nodes and edges compared to larger datasets, making it computationally efficient and suitable for quick experimentation. It is also widely studied in the literature, providing a solid baseline for comparison with established models.
4. **Rich Node Features:** Each paper is represented by a 1433-dimensional bag-of-words feature vector, providing GNN models with both topological (citations) and content-based data. This makes the dataset challenging and interesting for node classification.
5. **Small and Manageable Size:** Cora contains 2708 nodes and 10556 edges, making it small and manageable. This allows for fast iterations and testing of different models and hyperparameters without extensive computational resources.
6. **Transductive Learning Setup:** In Cora, the graph structure (features and edges) is known for all nodes, but only a subset of nodes is labelled during training. This transductive learning setup mirrors many real-world problems where node labels are not fully available but the graph structure is.

**Dataset Breakdown:**

```
Dataset: Cora():
Number of graphs: 1
Number of features: 1433
Number of classes: 7

Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708])
Number of nodes: 2708
Number of edges: 10556
Node feature dimension: 1433
Number of training nodes: 140
Is undirected: True
```

1. **Node Features (x):** Shape: [2708, 1433]

This means there are 2708 nodes (papers) in the graph, and each node has a 1433-dimensional feature vector. These features represent the content of each paper using a bag-of-words approach.

2. **Edge Index (edge_index):** Shape: [2, 10556]

The graph has 10556 directed edges, representing citations between papers. Since Cora is treated as an undirected graph, each citation is considered bidirectional (meaning every edge has two directions).

3. **Labels (y):** label:[7]

Each node (paper) belongs to one of 7 classes (research topics). The label $y_i$ indicates the class of node i.

4. **Train/Validation/Test Masks:** Shape: [2708]

The train_mask, val_mask, and test_mask are boolean masks used to indicate which nodes are part of the training, validation, and test sets.

Training nodes: 140 (used for training the model). Validation nodes: (used to tune hyperparameters and monitor model performance during training). Test nodes: (used to evaluate the model on unseen data).

5. **Is the graph undirected?**

Yes, the graph is treated as undirected. This means if paper A cites paper B, we assume paper B also "cites" paper A for the purposes of constructing the graph, doubling the edge count (hence, 10556 edges for 5429 directed citations).

**How did you pre-process the dataset?**

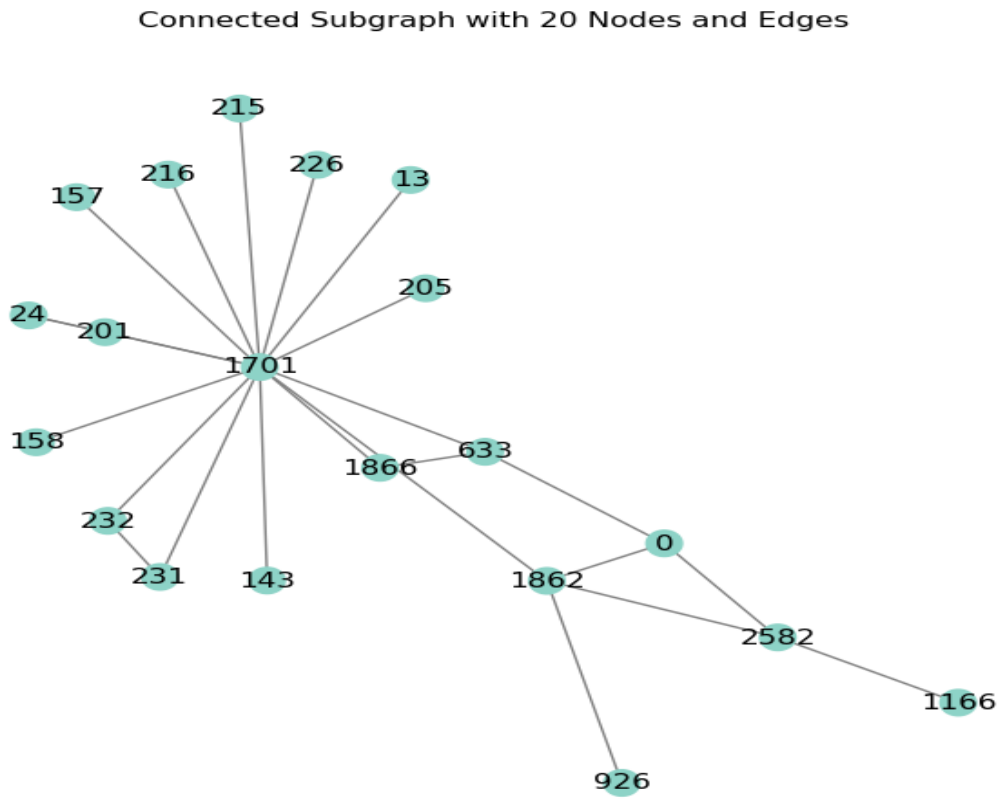
Pre-processing was performed to prepare the Cora dataset for GNN input. However, to ensure efficient training:

The main steps included:

1. **Data Transformation**: The dataset was transformed into a format compatible with PyTorch Geometric, with the graph data represented as Data(x, edge_index, y), where:
    - x is the node feature matrix (2708 nodes, 1433 features each).
    - edge_index holds the edges representing citations.
    - y contains the class labels for each node.
2. **Splitting Data**: The dataset was split into training, validation, and test sets using Boolean masks:
    - **Training set**: Contains a subset of labelled nodes for supervised training.
    - **Validation set**: Used to tune hyperparameters and monitor generalization.
    - **Test set**: Evaluate the model's performance on unseen data.

3. **Normalization**: **Batch normalization** will be applied in the model architecture after each GCN layer to standardize feature values and stabilize training.
4. **Computational Management**: Since the dataset size was manageable, additional computational reduction techniques (e.g., dimensionality reduction) were unnecessary. **Dropout** was used in the model architecture to prevent overfitting, adding regularisation and improving generalisation.

By combining feature normalisation, training masks, and dropout, the dataset will be prepared to efficiently leverage GNN architectures for the node classification task, ensuring balanced computation and generalisation.




This plot visualizes a connected subgraph of 20 nodes from the Cora dataset, where each node represents a research paper and edges represent citation relationships. All 20 nodes belong to Class 3, indicating that these papers share the same research topic. The nodes are coloured based on their class (research topic), with the subgraph selected using a Breadth-First Search (BFS) algorithm to ensure connectivity. The fact that all nodes in this subgraph belong to the same class emphasizes the strong citation relationships within this research topic. This could suggest a tightly knit cluster of papers, where authors heavily cite one another within the same field. Such visualization helps in understanding how closely related papers are organized in the Cora dataset and how GNNs can effectively leverage this structure for tasks like node classification

## SECTION 2

The model used in this project is DeepGCNWithBatchNorm, a custom Graph Convolutional Network (GCN) with batch normalization, designed specifically for node classification tasks like those in the Cora dataset. The architecture consists of multiple layers of GCNs, combined with batch normalization and dropout to improve training stability and prevent overfitting.

**Model Architecture:**

The model includes 4 GCN layers.

1. The first GCN layer takes the input features (1433-dimensional) and projects them into a 64-dimensional hidden space.
2. The next two intermediate layers retain the 64-dimensional hidden representation while aggregating neighbourhood information.
3. The final GCN layer outputs 7 dimensions, corresponding to the 7 classes in the Cora dataset (representing different research topics).

Each layer's transformation is represented by:

$$H^{(l+1)} = \sigma(\hat{A}\, H^{(l)} W^{(l)})$$

where:

- $H^{(l)}$ is the node feature matrix at the layer $l$,
- $W^{(l)}$ is the trainable weight matrix at layer $l$,
- $\hat{A}$ is the normalized adjacency matrix of the graph, and
- $\sigma$ represents the ReLU activation function.

```
DeepGCNWithBatchNorm(
  (convs): ModuleList(
    (0): GCNConv(1433, 64)
    (1-2): 2 x GCNConv(64, 64)
    (3): GCNConv(64, 7)
  )
  (bns): ModuleList(
    (0-2): 3 x BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

**Model Initialization:**

The model is initialized with an input size of 1433 (node features), a hidden layer size of 64, and an output layer of 7 (classes)

**Optimization and Loss Function:** The optimizer is Adam, with a learning rate of 0.01 and weight decay of $5 \times 10^{-4}$.The loss function is Negative Log-Likelihood (NLLLoss) as the model uses log-softmax activation for the output.

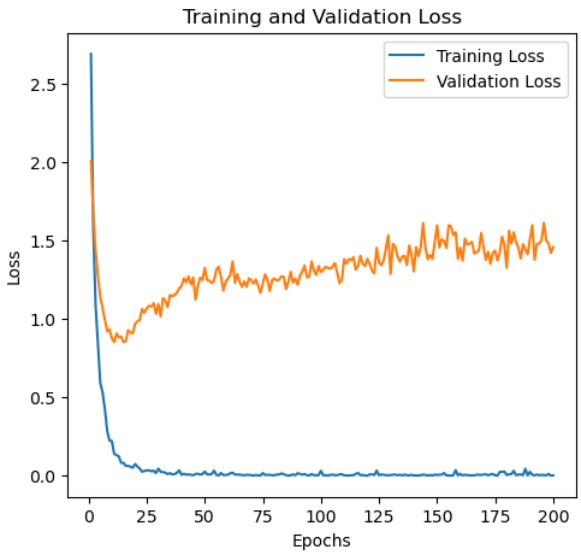
**Why do I choose the model?**

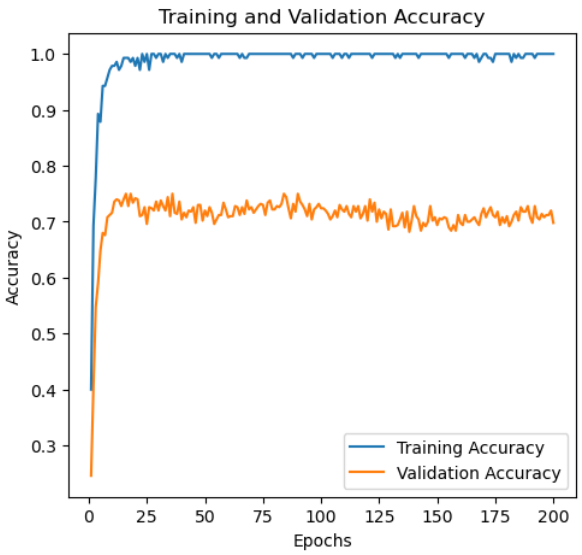
I chose GCN (Graph Convolutional Network) for this task primarily due to its simplicity, efficiency, and effectiveness in dealing with graph-structured data like the Cora dataset. GCNs are computationally lightweight compared to more complex models like GAT (Graph Attention Networks), making them a better fit for this project, where computational efficiency is important. GCN operates by aggregating features from a node's neighbours through a message-passing mechanism using the normalized adjacency matrix. This allows for capturing relationships between nodes without the need for learning attention weights, which GAT would require. Since the Cora dataset is relatively homogeneous, GCN's uniform aggregation approach is sufficient for the node classification task.

**Choice of Four Layers:** I selected a four-layer architecture to allow the model to aggregate information from nodes up to four hops away. This multi-hop aggregation enables each node's representation to reflect information from both its immediate and distant neighbors, essential for capturing relationships in the graph. Going beyond four layers might lead to over-smoothing, where node features become indistinguishable, reducing classification accuracy. Meanwhile, fewer layers might not capture enough contextual information from the graph.

**Batch Normalization and Dropout:** Batch normalization was included to stabilize the training process, allowing for faster convergence by normalizing feature values. Dropout was added to prevent overfitting, especially important in deeper architectures. This combination of batch normalization and dropout makes the model robust and better suited for generalization on unseen data.

# Section 3




**Key Observations:**

1. **Training Loss and Accuracy:**

The training loss decreases rapidly during the initial epochs and quickly stabilizes near zero, indicating the model is effectively fitting the training data. The training accuracy follows a similar trend, reaching close to 100% accuracy after about 25 epochs, suggesting that the model is performing almost perfectly on the training set.

2. **Validation Loss and Accuracy:**

The validation loss decreases initially but then starts to fluctuate after about 40 epochs, suggesting instability in the model's generalization ability. Around epoch 75, the validation loss shows considerable variation, and this trend continues throughout the remaining epochs. The validation accuracy also stabilizes early, hovering around 70-75%. Despite some fluctuations, it does not improve significantly after the first few epochs, further indicating that the model is struggling to generalize beyond the training data.

**Insights and Explanations**

The results indicate the following insights regarding the model's performance:

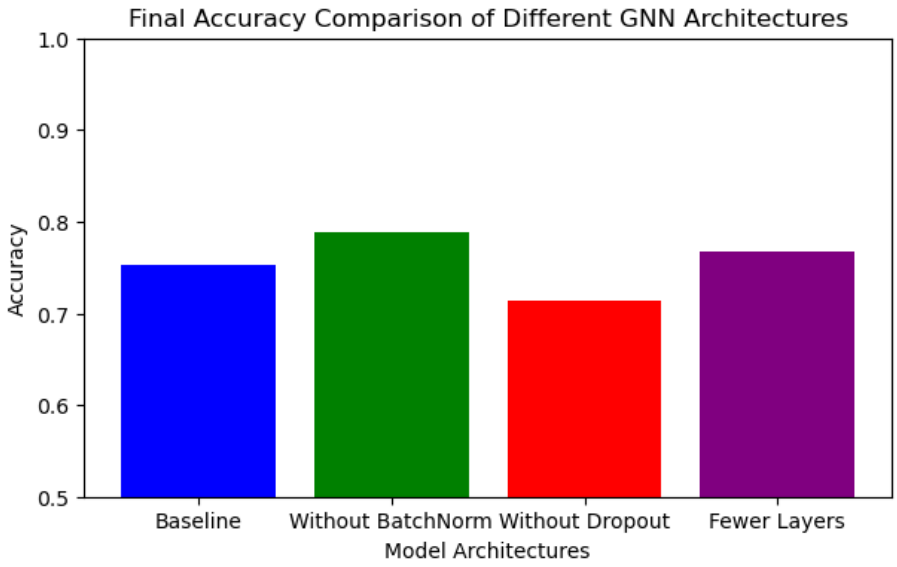
1. **Effectiveness of Batch Normalization and Dropout:**
    - Batch normalization stabilized the training process, enabling the model to converge quickly. However, it did not significantly enhance the validation accuracy. This might be because the relatively small size and uniform feature distribution of the Cora dataset do not benefit as much from batch normalization.
    - Dropout played a crucial role in preventing overfitting. The training accuracy approached 100%, while the validation accuracy plateaued, indicating that dropout effectively maintained generalization to some extent. Without dropout, the model likely would have overfitted even more severely.

2. **Signs of Overfitting:**
    - The gap between the training and validation accuracy curves, along with the increasing trend in validation loss, suggests overfitting. Although dropout helped reduce this, the high training accuracy and lower validation accuracy indicate that the model was overly specialized to the training data.
    - This overfitting could be due to the depth of the model (four layers), which allowed it to memorize training examples but made it harder to generalize. Reducing the model's complexity might help balance performance between training and validation.
3. **Potential Over-Smoothing in Deeper Layers:**
    - The model's four-layer depth enables it to aggregate information from distant neighbours, which is useful for capturing complex relationships. However, deeper layers in GCNs often lead to over-smoothing, where nodes from different classes become too similar, reducing the model's ability to distinguish between them. This could explain why the validation accuracy plateaued and even slightly declined in later epochs.

# Section 4

**Ablation Studies**

The ablation study systematically examines the impact of individual components in the Deep GCN with Batch Normalization and Dropout model by removing or modifying key elements in the architecture. Below are the results and insights from the ablation study:




1. Baseline Model Accuracy: 0.753

The baseline model includes batch normalization, dropout, and multiple layers. It achieves an accuracy of 0.753, which serves as the reference point for further comparisons. This model represents the original setup without any modifications.

2. No Batch Normalization Accuracy: 0.788

In this model, batch normalization was removed, and it surprisingly performs better than the baseline, achieving an accuracy of 0.788. This suggests that batch normalization might not be necessary for this dataset, and removing it allows the model to converge better.

3. No Dropout Accuracy: 0.714

Removing dropout from the model decreases the accuracy to 0.714, which is lower than the baseline. Dropout helps regularize the model and prevents overfitting, and this result shows that its absence negatively impacts the model's ability to generalize to unseen data.

4. Fewer Layers Accuracy: 0.767

Reducing the number of layers to 2 increases the accuracy to 0.767, which is slightly better than the baseline. This suggests that the model might perform better with a shallower architecture, potentially because deeper models overfit the dataset or have more difficulty generalizing.

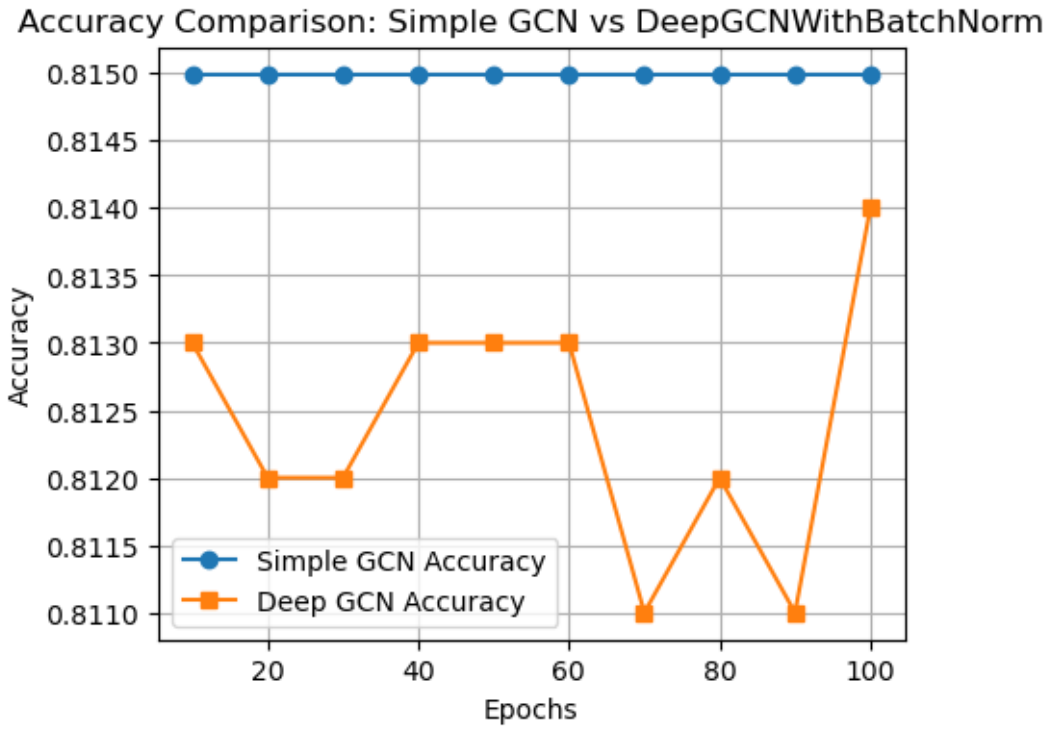
**Comparison Between Multiple Model Architectures**

To further understand the performance of the Deep GCN model, I compared it with a **Simple GCN** model and other modified versions. Below is a brief overview and the rationale behind each model selection:

1. **Deep GCN with BatchNorm and Dropout (Baseline)**:
    - The DeepGCNWithBatchNorm model is a more complex, custom-designed architecture with four GCN layers. In contrast to the Simple GCN model, this architecture incorporates batch normalization and dropout to improve stability and prevent overfitting during training. Batch normalization helps in faster convergence and regularization, making it a strong candidate for deeper networks.
2. **Simple GCN Model**:
    - The Simple GCN model, introduced in Week 7 practicals, is a straightforward two-layer GCN without batch normalization or dropout. It was selected as a baseline comparison to evaluate whether added complexity in the Deep GCN model provided any performance gains. As seen in the second graph, the Simple GCN model achieved a stable accuracy of around **0.815**, outperforming the Deep GCN model in terms of stability.

**Why Chosen**

I selected the Simple GCN as a baseline due to its simplicity and strong performance in many graph classification tasks. It's a well-known benchmark model and helps demonstrate the difference in performance when using a more complex architecture like the DeepGCNWithBatchNorm.



**Interpretation:**

In this graph, the accuracy of two models, Simple GCN and DeepGCNWithBatchNorm, is compared over 100 epochs. The key observations are as follows:

- **Simple GCN Model (Blue Line):** The accuracy of the Simple GCN model remains stable at around 0.815 throughout all the recorded epochs. This stability indicates that the model has quickly converged and does not experience significant variations in performance. It suggests that the Simple GCN model reaches its peak early and maintains that performance.
- **DeepGCNWithBatchNorm Model (Orange Line):** The Deep GCN model shows more fluctuations in accuracy compared to the Simple GCN model. The accuracy starts near 0.813, fluctuates between 0.811 and 0.813, and ends with an improvement near 0.815 at epoch 100. These fluctuations indicate that the model might be less stable during training, possibly due to its more complex architecture with batch normalization and multiple layers. This behaviour could be due to the model attempting to fine-tune and adjust over a longer period.

## Conclusion

The Simple GCN model outperformed the Deep GCN with Batch Normalization and Dropout in terms of stability and overall accuracy, demonstrating that simpler architectures can sometimes be more effective, especially on smaller datasets. The ablation studies showed that dropout is essential for maintaining generalization, while batch normalization and additional depth may not always contribute positively to model performance on such datasets.

These findings underscore the importance of selecting appropriate model complexity and regularization techniques based on the dataset characteristics. For future studies, exploring alternative architectures like Graph Attention Networks (GAT) or adjusting the depth and regularization further may yield additional insights. Overall, this study reaffirms that, while GNNs are powerful tools for graph-structured data, balancing model complexity with the dataset's specific needs is crucial for optimal performance.