Graph Neural Networks and Node Classification: Wiki-CS Case Study

Applied geometric deep learning on non-Euclidean data for node classification.

INTRODUCTION: Why study data on the graphs?

The shift toward Graph Neural Networks GNNs represents a transition from processing data as fixed, Euclidean arrays to treating it as dynamic, relational systems. While CNNs and Transformers are highly effective, they lack the architectural flexibility and have an inductive bias to navigate the irregular and complex relations of data in the real world. CNNs excel at capturing local spatial patterns but are inherently tied to fixed-grid layouts, making them incapable of handling the irregular, varying neighborhoods found in chemical compounds or social networks. Transformers, despite their flexibility, treat data as a bag of tokens where every element can potentially interact with every other. Graph Neural Networks were developed to bridge the gap between traditional deep learning and the complex, irregular structures found in the real world. They also handle permutation invariance because in models like MLP and CNN, they are more sensitive to the order of input. The neighbourhood message passing and dynamic neighbourhood excels the GNN learning capacity.

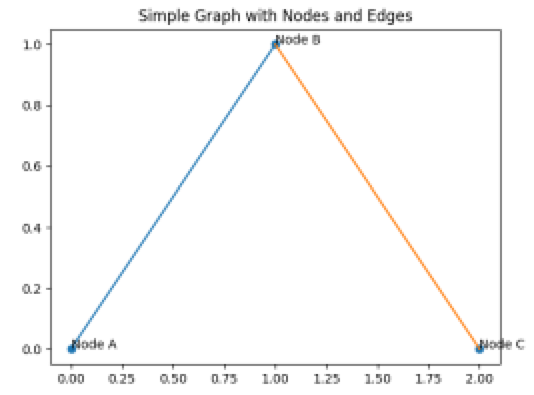
Why CNNs/MLPs are insufficient:

- No fixed grid structure in graphs
- Node ordering is fixed where graph needs permutative invariance.
- Neighborhood sizes vary.
- Topology and symmetry influence learning.

GRAPH STRUCTURE: Nodes and Edges

A Graph is a type of data structure that contains nodes and edges. A node can be a person, place, or thing, and the edges define the relationship between nodes. The edges can be directed and undirected based on directional dependencies.

Graph G = (V, E) V = set of nodes, E = set of edges
Adjacency Matrix A where A_ij = 1 if an edge exists, else 0



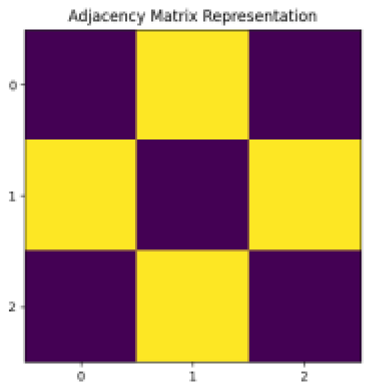


Figure 1: Graph and its Adjacency Matrix.

Graph Neural Network:

A Graph Neural Network are deep learning architecture that is used to learn from a graph structured data where traditional neural networks fail. GNNs are designed to operate on non-Euclidean data where the relationships and distances between points are irregular and flexible.

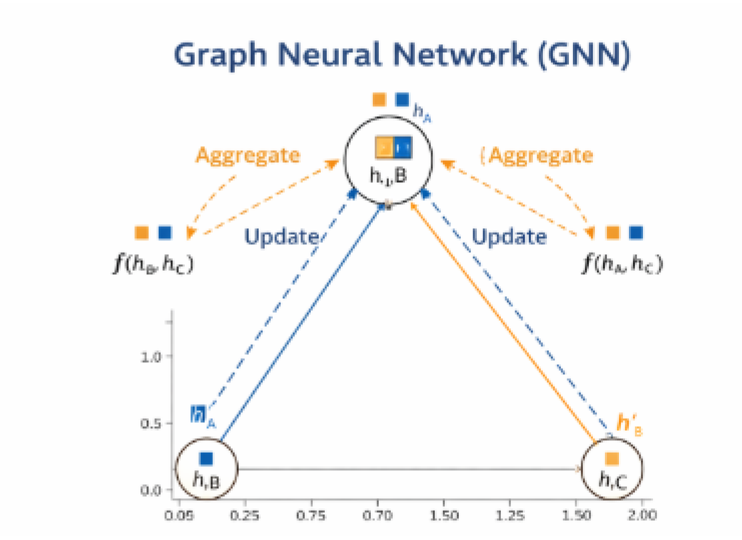


Figure 2: Graph Neural Network Architecture

Neighborhood Aggregation (Message Passing):

Each node in GNN updates its representation by aggregating features from its neighbors and updates its own state using a differentiable function. This allows the model to transfer information across the graph's non-Euclidean structure.

```
h_v^(k) = UPDATE^(k) ( h_v^(k-1), AGGREGATE^(k)({ h_u^(k-1) : u in N(v) }) )
```

Common aggregation methods:
1 Mean aggregation
2 Sum aggregation
3 Max pooling

Graph Convolutional Network:

A Graph Convolutional Network is an architecture that learns by having each node take an average of information from its direct neighbors to update its own internal state. It uses a specific normalization formula to ensure that nodes with many connections do not dominate the learning process, allowing the model to stay numerically stable while learning the graph's structure

```
H^(k+1) = sigma( D^(-1/2) A D^(-1/2) H^(k) W^(k) )
```

Where:
1) A = adjacency matrix
2) D = degree matrix
3) H = node feature matrix
4) W = learnable weights
5) sigma = activation function

Methodology:

1)Data Used:

- **Data:** Wiki-CS
- **Nodes:** Wikipedia CS articles, 11,701 nodes
- **Edges:** hyperlinks between articles 297,291 edges
- **Features:** 300-dimensional embeddings
- **Classes:** 10

2)Model Architecture

- **Architectures:** Implemented a deep Graph Convolutional Network (GCN) consisting of three sequential GCNConv layers

● **Layer Dimensionality:** 300-dimension input features, 64-dim latent embeddings, hidden refinement via neighborhood aggregation, 10-class output logits for node classification.

● **Residual Connections:** Incorporated a Skip Connection in the second layer to facilitate gradient flow and prevent the vanishing gradient problem.

● **Regularization:** Applied Dropout with a probability of 0.4 between each convolution layer to mitigate overfitting and improve the model's ability to generalize to unseen nodes

● **Non-Linear Activation:** Utilized the ReLU activation function after each convolution to introduce non-linearity into the feature transformation process.

● **Objective Function:** Automated Checkpointing and Early Stopping based on validation loss minimization.

● **Overfitting Mitigation:** Used softmax as the final activation to produce a probability distribution across the 10 categorical classes.

i) Model Training

● **Optimizer:** Employed the Adam optimizer with a learning rate of 0.01 for efficient stochastic gradient descent.

● **Weight Decay**: Applied a weight decay L2 regularization to the optimizer to prevent overfitting by penalizing large weights.

● **Loss Function**:**y:** Utilized the Negative Log-Likelihood Loss, which is the standard objective function for multi-class classification when paired with a Log-Softmax output.

● **Early Stopping Mechanism**: Implemented a patience-based early stopping strategy, the training process stops if the validation accuracy does not improve, preserving the best model state.

● **Evaluation Framework**: Evaluated by iterating through the dataset's training, validation, and test masks to calculate average performance metrics across different data segments.

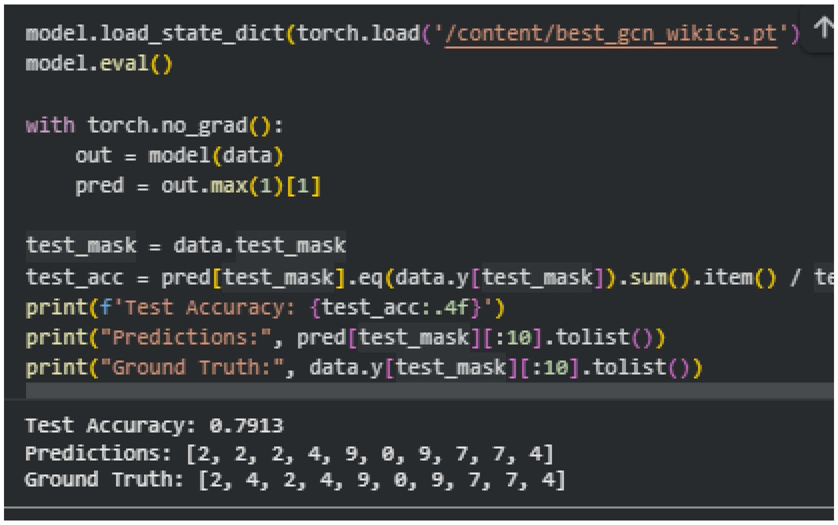
● **Efficiency & Persistence:** Final model performance was measured using Test Accuracy.

Evaluation_Metrices:

Epoch: 40 Loss: 0.4088 Train Accuracy: 0.8054 Test Accuracy: 0.7913 Val Accuracy: 0.8060

ii) Result

The Graph Convolutional Network achieved a final Test Accuracy of 79.13%, demonstrating strong generalization on unseen Wikipedia computer science articles.



```
model.load_state_dict(torch.load('/content/best_gcn_wikics.pt')
model.eval()

with torch.no_grad():
    out = model(data)
    pred = out.max(1)[1]

test_mask = data.test_mask
test_acc = pred[test_mask].eq(data.y[test_mask]).sum().item() / te
print(f'Test Accuracy: {test_acc:.4f}')
print("Predictions:", pred[test_mask][:10].tolist())
print("Ground Truth:", data.y[test_mask][:10].tolist())
```

```
Test Accuracy: 0.7913
Predictions: [2, 2, 2, 4, 9, 0, 9, 7, 7, 4]
Ground Truth: [2, 4, 2, 4, 9, 0, 9, 7, 7, 4]
```

Figure 3: Comparison of Predictions vs Ground Truth on the WikiCS Test Set

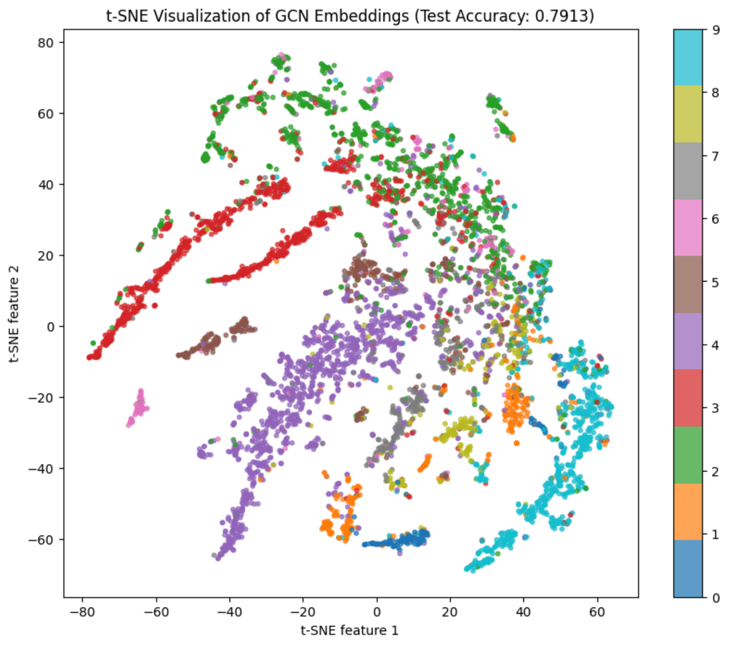


Figure 4: t-SNE visualization of the high-dimensional latent space

5) Conclusion :

In this research, a Graph Convolutional Network architecture was developed to classify interconnected Wikipedia articles, achieving a test accuracy of 79.13%. The model successfully identified categorical relationships within the WikiCS dataset with high precision, maintaining numerical stability. On a complex network of 297,291 hyperlinks, the GCN outperformed standard linear methods by effectively aggregating features from a node's local neighborhood and its own state. This success is attributed to the architecture's ability to capture structural dependencies across the graph, transforming 300-dimensional embeddings into highly accurate class predictions.