DGL 2026 Coursework 1

Introduction

Welcome to the first DGL coursework! In this coursework, you will explore advanced concepts in Graph Neural Networks (GNNs), moving beyond simple implementation to tackle topological reasoning, failure mode analysis, and principled architectural design.

Notebook Structure. This coursework is completed in a **single Jupyter Notebook**. You may reuse helper functions and variables across questions unless explicitly stated otherwise. However, your final submission must run correctly using `Restart Kernel & Run All`.

You will tackle three key challenges:

1. **Node Classification in Heterogeneous Spaces (40 Points):** You will work with a Drug Repurposing scenario where nodes exist in disjoint semantic spaces (Chemical vs. Protein). You must implement a **Cross-Modal Attention** mechanism from scratch to bridge this gap without relying on naive linear projections.

2. **Investigating Topology in Node-Based Classification (30 Points):** You will analyze how graph topology affects GNN performance. You will compare different graphs and analyze their spectral and topological properties.

3. **The Mystery Graph Investigation (30 Points):** You are provided with a dataset where standard GNN assumptions may not hold. You must discover the topological properties of this graph yourself, hypothesize why standard models fail, and design your own custom architecture to solve it.

Submission and Evaluation

- Explanations should be supported by evidence (e.g. plots, statistics, or controlled comparisons), not only intuition.
- Plots or metrics must be accompanied by an explanation of what they reveal and why this explains the model behavior.
- This coursework is completed in a single Jupyter notebook.
- You may reuse helper functions across questions unless explicitly stated otherwise.
- Your final submission must run correctly with Restart Kernel & Run All.

IMPORTANT NOTE: All implementations, explanations, discussions, plots, and figures must be included in your **Jupyter Notebook answers**.

1 Node Classification in Heterogeneous Spaces (40 points)

In this section, you are given a **bipartite** graph with two node types:

1. **Drugs:** chemical fingerprints (sparse vectors)
2. **Proteins:** sequence embeddings (dense vectors)

The feature spaces are **disjoint** (different semantics, different dimensions). You must predict **drug labels**.

Real-World Scenario: Drug Repurposing

Imagine a biomedical knowledge graph used for drug discovery. We have two types of nodes:

1. **Drugs ($u \in \mathcal{U}$):** Represented by chemical structure fingerprints (e.g., Morgan fingerprints, dimension d_1).
2. **Proteins ($v \in \mathcal{V}$):** Represented by amino-acid sequence embeddings (e.g., ProtBERT embeddings, dimension d_2).

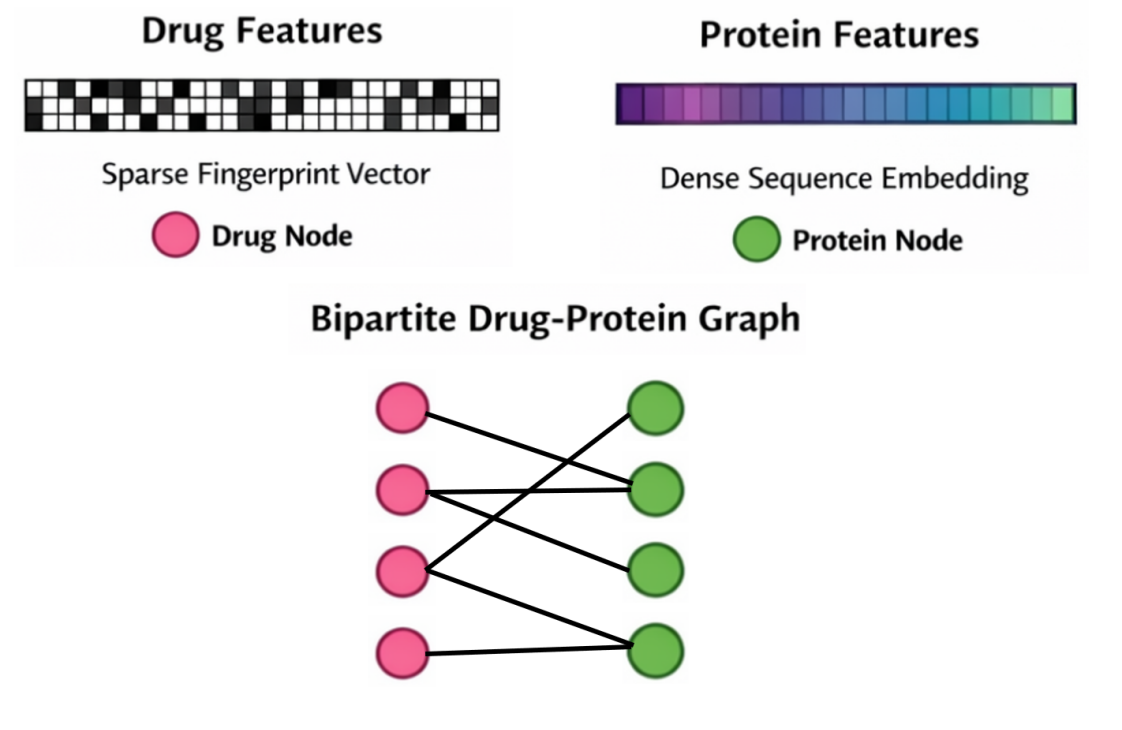

Figure 1: A demonstration of the drug and protein features and bipartite graph.

The Challenge: The features of drugs and proteins exist in completely disjoint semantic spaces. A vector in the chemical space cannot be directly compared or summed with a vector in the protein space. Furthermore, the graph is bipartite: Drugs only connect to Proteins, and Proteins only connect to Drugs.

Rules

1. **Allowed:** Python, NumPy, PyTorch, matplotlib, scikit-learn, seaborn
2. **Not allowed:** DGL, PyG, or any prebuilt GNN/attention layers

1.1 Dataset Analysis (10 Points)

1.1.a Exploratory Data Analysis (2 pts)

Task: Calculate the following statistics to understand the "Semantic Gap":

1. The **sparsity** (percentage of zeros) of the Drug features.
2. The **mean and standard deviation** of the Protein features.
3. The **average degree** (number of protein neighbors) per drug.

1.1.b Feature Density Analysis (2 pts)

A critical part of working with heterogeneous graphs is understanding the numerical distribution of different node types. **Task:** Implement the `plot_feature_density` function to visualize the distribution of average feature values for all Drugs versus all Proteins.

1.1.c t-SNE Gap Visualization (2 pts)

Task: Use t-SNE to project both drugs and proteins into a 2D space. **Hint:** Drug and protein features have different dimensions and cannot be concatenated directly. You must first project them into the same dimension using a fixed random projection (e.g., 128 dimensions) before running t-SNE.

1.1.d Graph Visualization (2 pts)

You must visualize **a small part** of the bipartite graph:

1. Sample `n_drug_sample` drugs.
2. Include their connected proteins.
3. Draw the bipartite subgraph.

Your plot should make the bipartite structure obvious (two vertical columns: drugs on left, proteins on right).

1.1.e Theoretical analysis (2 pts)

Consider a naive approach: project drugs and proteins into the same hidden dimension with linear layers, then mean/sum aggregate. Explain why this is suboptimal in this bipartite setting, referring to:

1. **Over-smoothing / signal dilution**
2. **Manifold alignment**

1.2 Cross-Modal Attention (CMA) (25 Points)

You will implement a custom Cross-Modal Attention (CMA) layer to update drug node representations using information from their protein neighbors. Because the graph is bipartite and feature spaces are disjoint, we treat the **Drug** as the Query and the **Proteins** as Keys and Values.

Mathematical Formulation: The updated representation for a drug node u is calculated using a residual connection:

$$h_u^{(l+1)} = \sigma\left(\sum_{v \in N(u)} \alpha_{uv} W_{val} h_v^{(l)} + h_u^{(l)}\right)$$

The attention coefficient α_{uv} is computed via a neighborhood-specific softmax:

$$\alpha_{uv} = \text{softmax}_{v \in N(u)}\left(\text{LeakyReLU}\left(a^T [W_q h_u^{(l)} \| W_k h_v^{(l)}]\right)\right)$$

1.2.a Segment softmax helper (5 pts)

Implement the `segment_softmax` function. Attention must be a probability distribution **per node neighborhood**. *Note:* To ensure numerical stability, use the "Log-Sum-Exp" trick or subtract the maximum value before exponentiating. The softmax must be normalized strictly over the neighbors of each individual drug using the `ptr_drug` array.

1.2.b Cross-Modal Attention Layer (5 pts)

Implement the `CrossModalAttention` module using basic PyTorch operations.

1. **Shared Latent Space:** Project inputs to shared d_{hidden}.
2. **Directional Processing:** Drugs as Queries (Q), Proteins as Keys (K) and Values (V).
3. **Message Aggregation:** Aggregate protein values weighted by attention.
4. **Residual Connection:** Add aggregated info back to projected drug representation.

1.2.c Discussion (10 pts)

Since the task is drug classification but the graph is bipartite, why is a 1-layer model structurally incapable of exploiting drug–drug information for this task? Explain mathematically.

1.2.d Heterogeneous CMA Classifier (5 pts)

We recommend a **2-step** update to allow information flow via drug→protein→drug (2-hop paths). You will implement:

- Drug classification model using CMA layers.
- Protein update from drugs.
- Drug update from proteins.

1.3 Alignment Loss (5 Points)

You must implement:

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \lambda \mathcal{L}_{align}$$

Where $\mathcal{L}_{align}$ pulls **connected** drug-protein pairs closer in a **shared latent space**.

1.3.a Implementation (2 pts)

Implement `alignment_loss(...)` for connected edges only. Define z_d and z_p as the d_{hidden}-dimensional embeddings produced by your model (after projection into the shared latent space).

1.3.b Analysis (3 pts)

Compare $\lambda = 0$ vs $\lambda > 0$. Plot F1 curves and discuss:

1. Did alignment help convergence speed? Why?
2. Did it improve peak validation F1?
3. Did it over-constrain the space?

2 Investigating Topology in Node-Based Classification Using GNNs (30 points)

In this section, we will explore the impact of graph topology on node-based classification using GNNs. The experiments will focus on analyzing different topological measures, visualizing their distributions, and evaluating GCN performance on 2 graphs with different topologies.

2.1 Analyzing the Graphs (10 points)

2.1.a Topological and Geometric Measures (3 pts)

- Examine two topological measures (**Node Degree** and **Betweenness Centrality**) and one geometric measure (**Ollivier-Ricci Curvature**).
- Include a definition for each measure.
- Explain **mathematically** and **intuitively** what each measure quantifies and how it contributes to understanding the structure of a graph.

2.1.b Visualizing and Comparing Topological and Geometric Measures (3 pts)

Implement the following functions to visualize and compare topological properties of two given graphs:

- `plot_node_degree_distribution_two_graphs`
- `plot_betweenness_distribution_two_graphs`
- `plot_ricci_curvature_distribution_two_graphs`

Discussion: Compare the topologies of two graphs using the topological and geometrical measures that you computed. What are your observations?

2.1.c Visualizing the Graphs (2 pts)

Implement the `plot_graph` function to generate visual representations of both graphs. Comment on the topologies.

2.1.d Visualizing Node Feature Distributions (2 pts)

- Implement the function `plot_node_feature_dist_by_class_two_graphs` to visualize the average node feature distribution per class for two given graphs.
- Do not consider the distribution of a specific feature x_i, consider the mean of the feature vector $\mathbf{x}$ for each node.
- **Discussion:** Analyze the results. Discuss any observed overlaps between the node feature distributions across different classes. Are the features well-separated, or do they exhibit significant overlap? Compare the distributions between the two graphs.

2.2 Evaluating GCN Performance on Different Graph Structures (10 points)

2.2.a Implementation of Layered GCN (1 pt)

- Modify the `GraphNeuralNetwork` class so that `num_layers` can be passed as an argument.
- Ensure it returns all embedding layers if requested (for t-SNE analysis).
- Train on G1 and G2 independently and plot results.

2.2.b Plotting t-SNE Embeddings (1 pt)

Use t-SNE to visualize the node embeddings from the **final hidden layer** of the GCN.

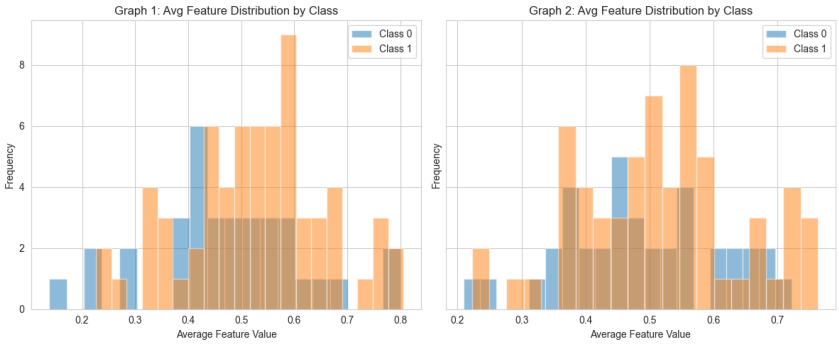

Figure 2: Example of node feature distribution per class.

2.2.c Training on Merged Graphs (2 pts)

- Complete `unify_two_graphs` to create $G_{union} = G_1 \cup G_2$.
- Train the GCN on this unified graph.

2.2.d Merged vs. Independent Training (6 pts)

Analyze and compare the training on two graphs independently versus training on their union. Your analysis must include both empirical evidence and conceptual reasoning.

Required Analysis:

1. **Training behaviour:** Compare convergence trends, stability, and speed.
2. **Learned representations:** Compare node embeddings obtained from independent vs. merged training.
3. **Overall performance comparison:** Summarise differences in predictive performance and representation quality.

Hypothesis & Trade-offs: Formulate a hypothesis explaining why observed differences occur (referencing graph structure and information mixing). Discuss advantages and drawbacks (computational, modeling, practical) and propose modifications that could mitigate drawbacks.

Bonus (Optional): Implement your proposal for learning on G_{union} in a better way.

2.3 Topological Changes to Improve Training (10 points)

2.3.a Plot the Ricci Curvature for each edge (1 pt)

Create a bar plot showing the Ricci curvature for each edge. The x-axis should represent the edges, and the y-axis should represent their corresponding Ricci curvature values.

2.3.b Extreme Topological Scenarios (4 points)

Explore limit cases of graph topology for both G1 and G2:

1. **Topology removal:** Construct a version of the graph where neighborhood aggregation becomes meaningless (behaves like MLP).
2. **Oracle topology:** Construct an idealized graph assuming label information is available (only for analysis).
3. **Analysis:** Discuss what these extreme cases reveal about the relationship between graph topology and node labels.

2.3.c Graph Augmentation Without Labels (5 pts)

Design a practical graph augmentation strategy that could be applied **without access to labels**.

- Implement `augment_graph` using only node features X and/or original topology A.
- **Interpretation:** Explain your methodology, what assumptions it encodes, and why it helps (or fails) on this dataset. Support with quantitative results.

3 Mystery Graph Investigation (30 pts)

In this question, you will work with a **mystery graph dataset** and investigate the behavior of graph-based learning. You are not told anything about the structure of the dataset in advance. Your task is to *discover* what is happening and respond accordingly.

Rules

1. Allowed: Python, NumPy, PyTorch, matplotlib, scikit-learn.
2. Do not modify the dataset.

3.1 Baseline Models

You will first train two baseline models:

1. **MLP:** ignores the graph completely.
2. **GCN:** uses mean aggregation over neighbors.

At this stage, **do not try to improve anything**. Simply train both models and observe their performance.

3.2 Explaining the Results (15 pts)

You should observe that **the MLP performs better than the GCN**. This is not what one might expect if the graph were always helpful.

Task: Using **quantitative measures and visualisations**, explain:

1. Why does the GCN perform worse than a feature-only MLP? Considering a GCN does neighborhood aggregation, it is not expected an MLP to perform better than a GCN. Explain this phenomenon.
2. What is the relationship between node labels and the graph structure?
3. What kind of information does **mean aggregation** preserve or destroy here?
4. How is this phenomenon related to homophily in graphs and GNNs?

3.3 Designing a Better Graph Method (15 pts)

The GCN fails because of *how* it aggregates information from neighbors. Your task is to design a **new message passing method** that uses the graph but avoids the failure mode of mean aggregation on this dataset.

Requirements:

- Address the failure reason that you discovered in Stage 2.
- Be sophisticated rather than simply adding more layers and increasing number of learnable parameters.
- If your reasoning is about the optimization strategy rather than the architecture, you need to apply your new optimization strategy to classical GCN and MLP too in order to be fair.

Final Explanation: Describe what failure mode you identified, how your method addresses it, why it improves performance, and what assumptions your method makes about the graph.

4 Submission Instructions

Your coursework submission must be in the form of a **ZIP file** containing the following:

1. The completed Jupyter Notebook with **generated cell outputs**.
2. **No separate PDF report is required.** Type your answers directly in the Jupyter Notebook using Markdown.
3. The data folder must remain the same as provided.
4. `requirements.txt` must be updated if additional libraries are used.

4.1 File Organization

```
submission.zip
|-- notebook.ipynb
|-- data/ (folder same as provided)
|-- requirements.txt
```

4.2 Penalties

1. **Logs not displaying:** Ensure logging info is shown.
2. **Uncommented code:** Explain your code blocks.
3. **Non-executable code:** All cells must run without error.
4. **Missing citations:** Cite external sources properly.