Programming Assignment 1: GNN-based Node Classification on Cora

Summary of Findings for Graph Learning (2025W)

Simon Schindler, Marlene Grabner, Aryan Rahbari

December 15, 2025

1 Introduction

This report summarizes the findings for the semi-supervised node classification tasks on the Cora citation network, as required by Programming Assignment 1. We utilized the PyTorch Geometric library for all experiments.

Python Environment Setup

To run the Jupyter notebooks and reproduce the results, a specific Python environment is required. The following steps outline the setup process using `uv`, a fast Python package installer.

1. **Install `uv`:** If not already installed, run the following command:

   ```
   curl -Lsf https://astral.sh/uv/install.sh | sh
   ```

2. **Create Virtual Environment:** This project uses Python 3.13. From the project root, create a virtual environment:

   ```
   uv venv
   ```

   `uv` will automatically detect the required Python version from the `.python-version` file.

3. **Install Dependencies:** Activate the environment and install packages using the `sync` command:

   ```
   uv pip sync
   ```

   This command uses the `pyproject.toml` and `uv.lock` files to ensure a reproducible environment. You can then start Jupyter Lab by running `jupyter lab`.

2 Task 2(a): Hyperparameter Optimization and Best Model

The objective of this task was to find a GCN-based model that achieves at least 80% test accuracy on the Cora dataset using the standard validation split. A comprehensive grid search was performed over key hyperparameters, including hidden dimensions, number of layers, dropout rate, learning rate (LR), and convolutional layer type (GCNConv, GraphConv, GATConv).

Table 1 summarizes the hyperparameters evaluated.

Hyperparameter	**Evaluated Values**
Hidden Dimensions	**16**, 64, 128
Number of Layers	**2**, 4, 8
Dropout	0.3, **0.5**, 0.7
Learning Rate	0.01, 0.005, **0.001**
Epochs	10, 50, **200**
Convolution Layer	**GCNConv**, GraphConv, GATConv

Table 1: Hyperparameters evaluated in the grid search. The best configuration is marked in **bold**.

2.1 Optimal GNN Architecture

The best-performing model, selected based on the highest validation accuracy, significantly exceeded the target, achieving a Test Accuracy of 88.50%. The optimal hyperparameters and resulting performance metrics are summarized in Table 2.

Table 2: Optimal GCN Architecture and Performance on Cora

Hyperparameter	**Setting**	**Notes**
Convolutional Layer	`GCNConv`	Standard GCN
Number of Layers	2	Shallow architecture
Hidden Channels	16	Smallest dimension tested
Dropout Rate	0.5	Optimal regularization
Learning Rate (LR)	0.001	Slow, stable convergence
Epochs	200	Full training run
Performance	**Accuracy**	
Training Accuracy	95.36%	
Validation Accuracy	88.00%	(Used for selection)
Test Accuracy	88.50%	($>$ **80**% target reached)

2.2 Key Hyperparameter Influences

Analysis of the grid search results revealed clear patterns regarding model performance and overfitting:

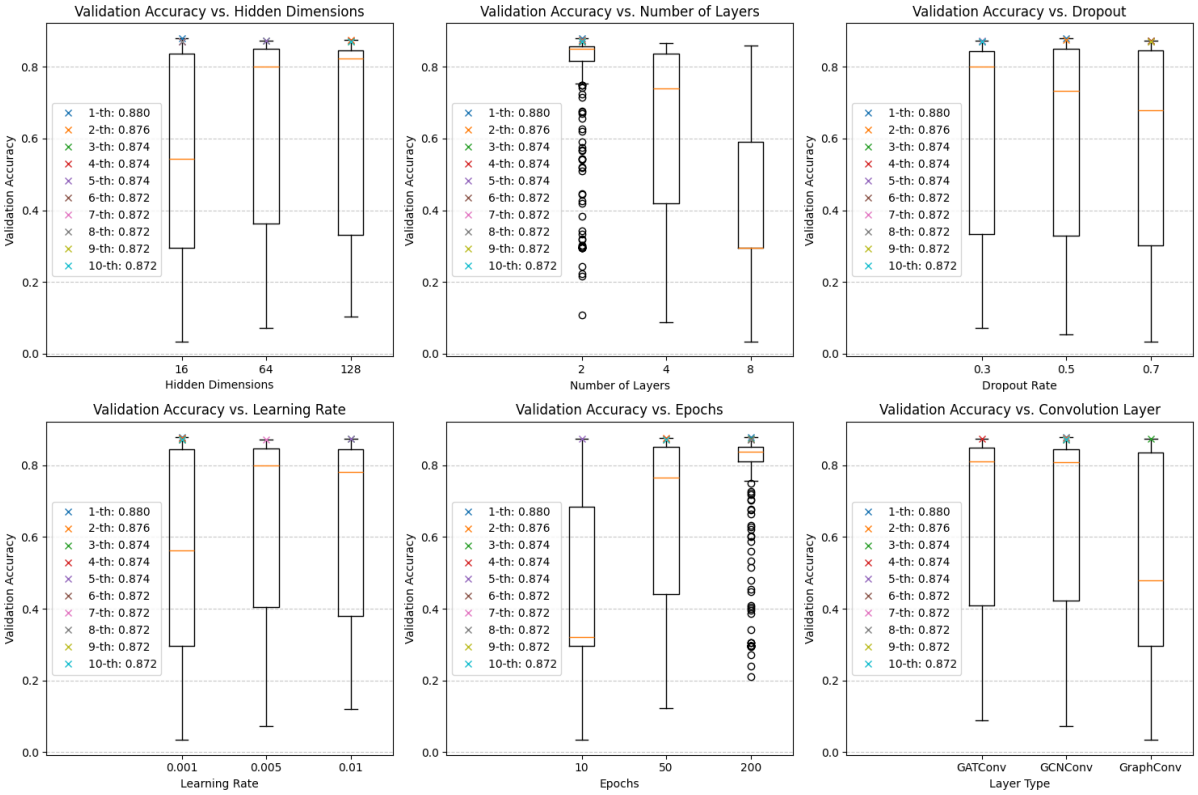

Figure 1: Influence of model hyperparameters on validation accuracy. Best 10 runs are individually marked with crosses.

Analyzing fig. 1 yields the following insights:

- **Number of Layers:** Performance peaked dramatically at 2 layers. Deeper models (4 and 8 layers) showed a significant drop in accuracy, indicating that the standard GCN architecture quickly suffers from **oversmoothing** on this dataset, where node representations become indistinguishable across the graph.
- **Hidden Dimensions:** The smallest dimension tested, 16, was optimal. Models with 64 or 128 channels showed decreased accuracy and a greater tendency toward overfitting (larger gap between training and validation accuracy).
- **Learning Rate:** A low LR of 0.001 was crucial for achieving the highest accuracy, suggesting that fine-tuning weights over many steps (200 epochs) is more effective than rapid learning.
- **Layer Type:** The standard `GCNConv` performed marginally better than `GraphConv` and significantly better than `GATConv`, suggesting the attention mechanism was not necessary for performance improvement in this setting.

3 Task 2(b): Homophily Analysis

This task focused on analyzing the graph's homophily and investigating the hypothesis that nodes with low homophily scores are more likely to be misclassified by the optimized GNN.

3.1 Homophily Measurement

3.1.1 Overall Homophily

The overall Node Homophily H_{node} was calculated using the formula:

$$H_{\text{node}} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \frac{|\{(w, v) : w \in \mathcal{N}(v) \wedge y_v = y_w\}|}{|\mathcal{N}(v)|}$$

The computed homophily score for the Cora dataset is $\mathbf{H}_{\text{node}} \approx \mathbf{0.8252}$. This confirms that Cora is a **highly homophilous graph**, where approximately 82.5% of a node's neighbors share its class label.

3.1.2 Homophily Distribution

Figure /reffig:homohist visualizes a histogram of per-node homophily scores showing a severely skewed distribution, with the vast majority of nodes (over 65%) residing in the highest homophily bin $(0.9 - 1.0)$, further solidifying the homophilous nature of the graph.

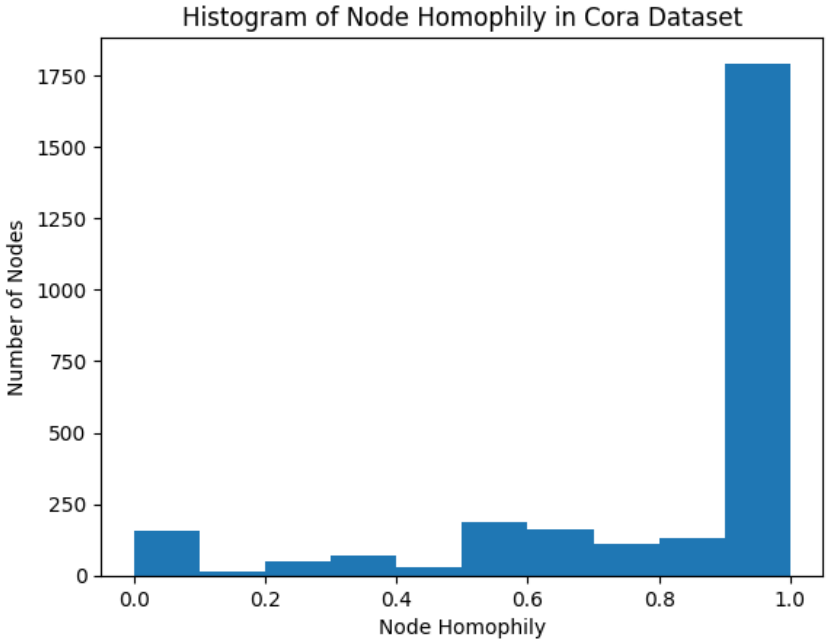

Figure 2: Homophily Distribution in Cora

3.2 Predictive Performance vs. Homophily

3.2.1 Experimental Design

To test the relationship between homophily and predictive performance, the nodes in the validation set were partitioned into 10 bins based on their individual homophily score (e.g., $0.0 - 0.1$, $0.1 - 0.2$, ..., $0.9 - 1.0$). The classification accuracy of the top 10 models (identified in Task 2a) was then measured for the nodes within each bin.

3.2.2 Findings

The results, visualized in a box plot (Figure 3), provide a clear answer to the central question:

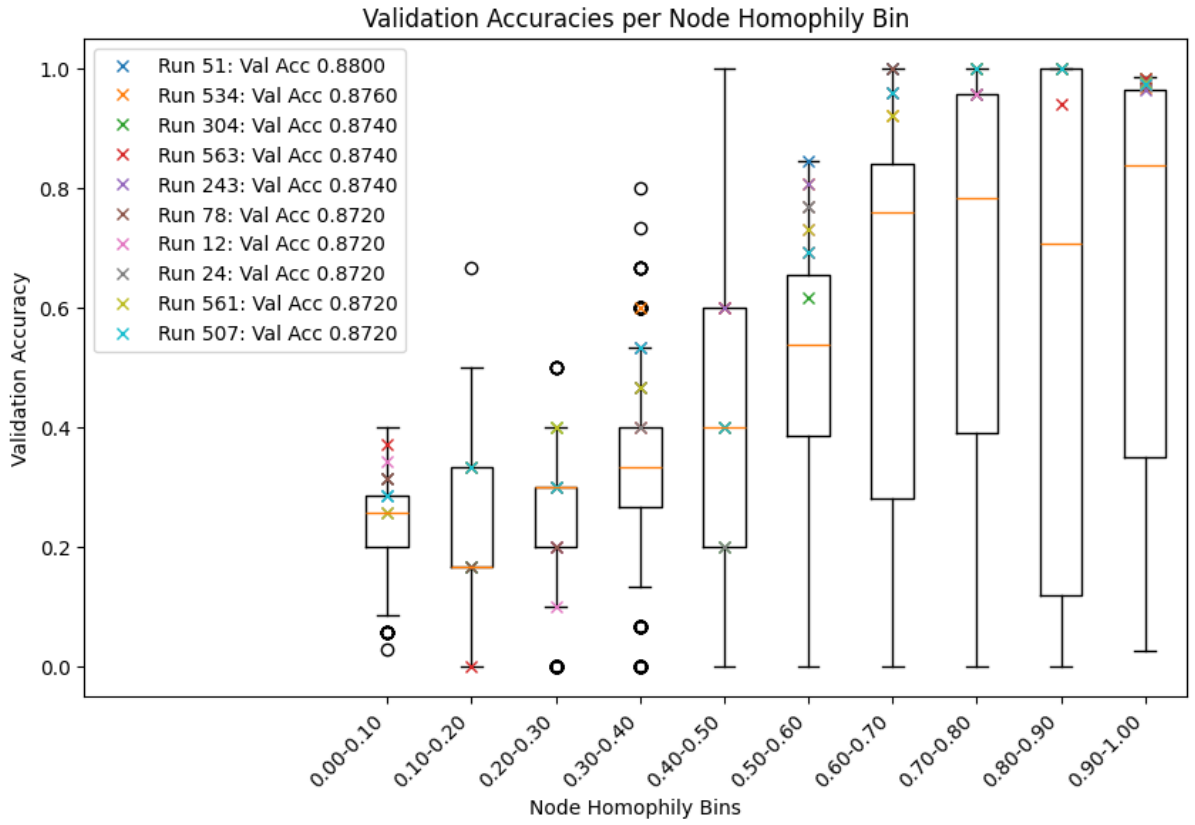

Figure 3: Validation Accuracy Distribution Across Node Homophily Bins

- **High Homophily (≥ 0.8):** Models show **excellent performance**. In the $0.9 - 1.0$ bin, the median accuracy across all runs approached 100%. The GCN performs optimally when the neighborhood signal is consistent.
- **Low Homophily (< 0.5):** The models consistently struggle. The median accuracy in the $0.0 - 0.1$ bin is very low (well below 20%), indicating that the majority of nodes in this bin are misclassified.
- **Conclusion:** Nodes with low homophily are **significantly more likely to be misclassified**. This is a characteristic failure mode of standard GCNs, which operate under the assumption of homophily. When a node is heterophilous (linked to different classes), the message-passing mechanism aggregates conflicting labels, corrupting the node's representation and leading to incorrect predictions.

4 Task 2(c): Influence of Number of Convolutional Layers

Oversmoothing is an effect in Graph neural networks where, with an increasing number of layers, node representations become increasingly similar. This removal of local information leads to the model no longer being able to distinguish between nodes, causing a drop in predictive performance. Mathematically, this can be compared to a low-pass filter, where high frequency information (local differences) is continuously removed until embeddings converge to a global average.

To see the phenomenon of oversmoothing, we used the previously found best hyperparameters and varied only the number of convolutional layers in the GCN from 2 to 128. We recorded the test accuracy and the Dirichlet energy of the node features after each layer.

4.1 Accuracy vs. Number of Layers

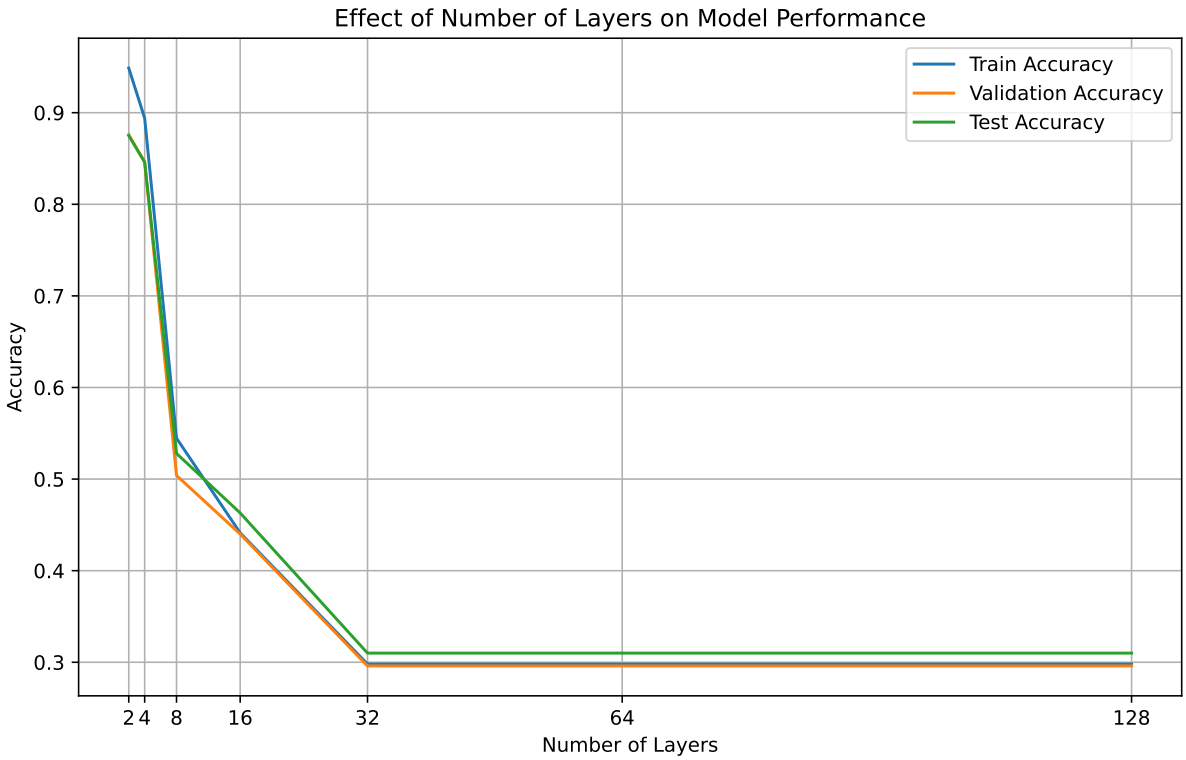

Figure 4: Effect of Number of Layers on Model Performance

Figure 4 shows the achieved accuracy on the task against the number of layers a model contains. With shallow models we observe peak performance (around 90 % accuracy). Here, the receptive field is large enough to capture local context, but unique node features are still preserved. With increasing number of layers, we observe a strong drop in accuracy. This happens, because the receptive field of the model exceeds the diameter of the graph. The Cora dataset is a small world network and has a diameter of 19 in its largest connected component. A 32-layer GNN on the other hand, expands the receptive field of a node to 32 hops. This means that every single node is aggregating information from every other node in the graph multiple times. The node embeddings therefore trend towards a global average of the entire dataset. Unique local features, which are required for classification are washed out. With the Cora Dataset having a high homophily, this local information is particularly relevant.

4.2 Dirichlet Energy of Layer Outputs

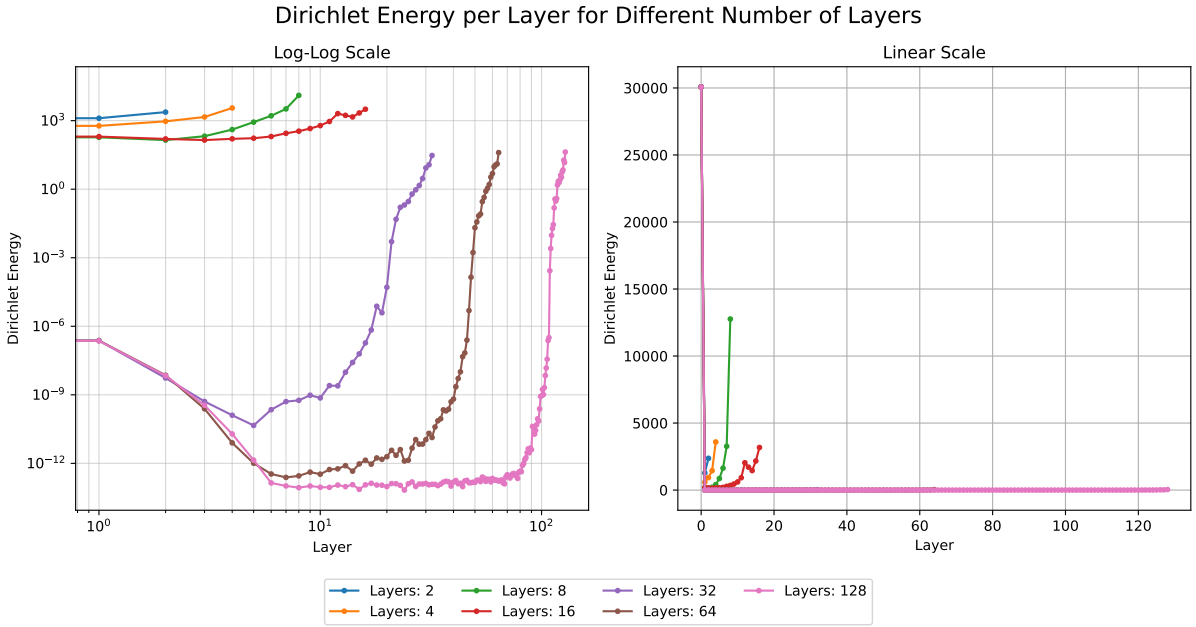

Figure 5: Dirichlet Energy per Layer for Different Number of Layers

Figure 5 shows the values of the Dirichlet Energy for the output of each layer of a model, against the number of the layer. This allows a layer-wise perspective on the smoothing process. On the left the results can be seen on a log-log scale, on the right on a linear scale. As seen in the linear plot, all models begin with the same high Dirichlet energy at layer 0. This represents the high variance in the input features. The log-log plot then shows a strong decay for the deepest models. There, the energy decreases to close to zero within the first ten layers (especially drastic in the 64 and 128 layer networks). In this zone the hidden representations are effectively all identical, indicating that subsequent layers are computationally useless.

In the final layers, we see that the Dirichlet energy for the deep networks rises again. This happens because Cross-Entropy Loss heavily penalizes predicting uniform probabilities (=low confidence). However, since the network is learning from oversmoothed and uninformative node representations at this point, this forces the final layer to magnify tiny, random numerical differences into large score differences. This artificial increase in Dirichlet energy is not related to the actual capture of information, which is also evident by the low prediction accuracy for these types of networks seen in figure 4.

5 Task 2(d): Effect of label change on oversmoothing

The goal of this task was to investigate if oversmoothing is a mechanical artifact of GNNs or if it is also influenced by the learning objective. To do this we replaced the original labels with a structure based label - the node degree.

This transformed the task from a classification to a regression task, where the model had to predict the degree of each node. To achieve this we replaced the Cross-Entropy loss with a Mean Squared Error loss and removed the final softmax activation to allow the prediction of continuous values.

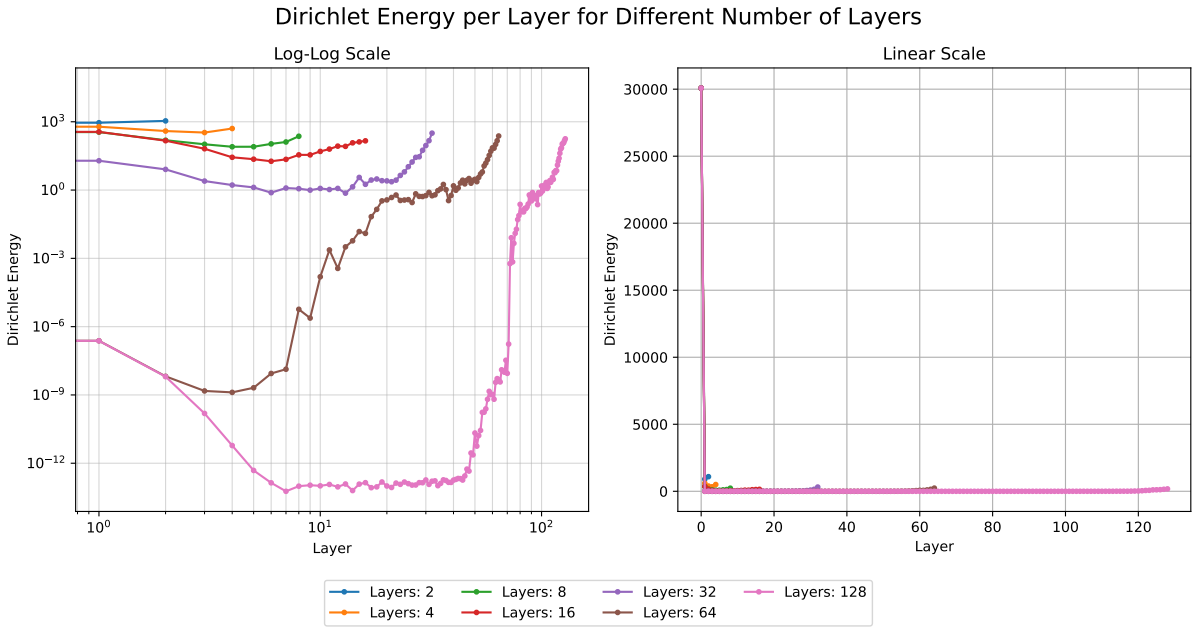

Figure 6: Dirichlet Energy per Layer for Different Number of Layers when predicting Node Degree

Figure 6 shows the results for the Dirichlet energy per layer for all the different layers of every tested model. We find that in comparison to the previous task, particularly the 32 layer model does not show an as strong decay in Dirichlet energy.

These results indicate that oversmoothing is not solely a mechanical artifact of GNNs, but also influenced by the learning objective. In the original task, the high homophily of the graph meant that nodes with similar features also had similar labels. As a result, the GNN was incentivized to smooth node representations to capture this similarity, leading to oversmoothing. In contrast, predicting node degree does not benefit as much from smoothing, as nodes with similar degrees are not necessarily closely connected.

Nevertheless, we still observe a decay in Dirichlet energy for deeper models, indicating that oversmoothing is still present. This suggests that while the learning objective influences the degree of oversmoothing, the architectural properties of GNNs inherently lead to some level of smoothing as layers are added [1].

6 Task 2(e): Oversmoothing and Activation Functions

The goal of this task was to investigate the influence of different activation functions on the oversmoothing phenomenon. The following activation functions were tested: Linear, ReLU and Sigmoid.

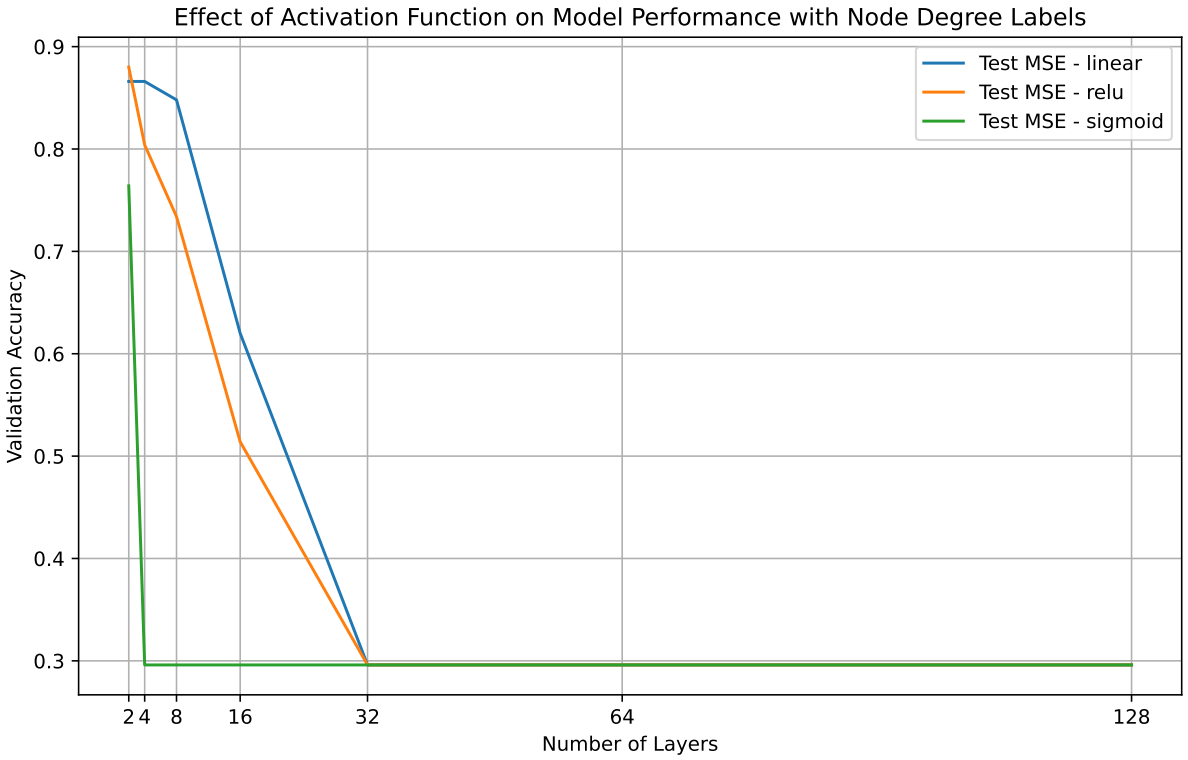

Figure 7: Effect of Activation Function on Model Performance

Figure 7 shows the achieved accuracy on the task against the number of layers a model contains for the different activation functions. We observe that the Sigmoid activation function leads to the worst performance overall, as well as the quickest drop in accuracy. There the accuracy is already only 0.3 for the 4 layer model.

Accuracy is similar for the 2-layer models using ReLU and Linear activation functions. However, with increasing number of layers the linear activation minimally outperforms the ReLU activation, before they both drop to an accuracy of 0.3 at 32 layers.

The poor performance of the sigmoid activation function can be explained by the vanishing gradient problem. The sigmoid function squashes any input value into the range between 0 and 1. Large absolute values are therefore mapped close to these borders. There, the gradient of the sigmoid function is small, leading to very small weight updates during backpropagation. Additionally, this effect gets even worse for multiple layers, where the gradients are multiplied for each layer during backpropagation.

The slightly better performance of the linear activation function compared to the ReLU activation is likely due to the nature of the Cora dataset. The high homophily in the dataset means that simply taking the average of the neighbors can be a successful strategy to achieve correct classifications.

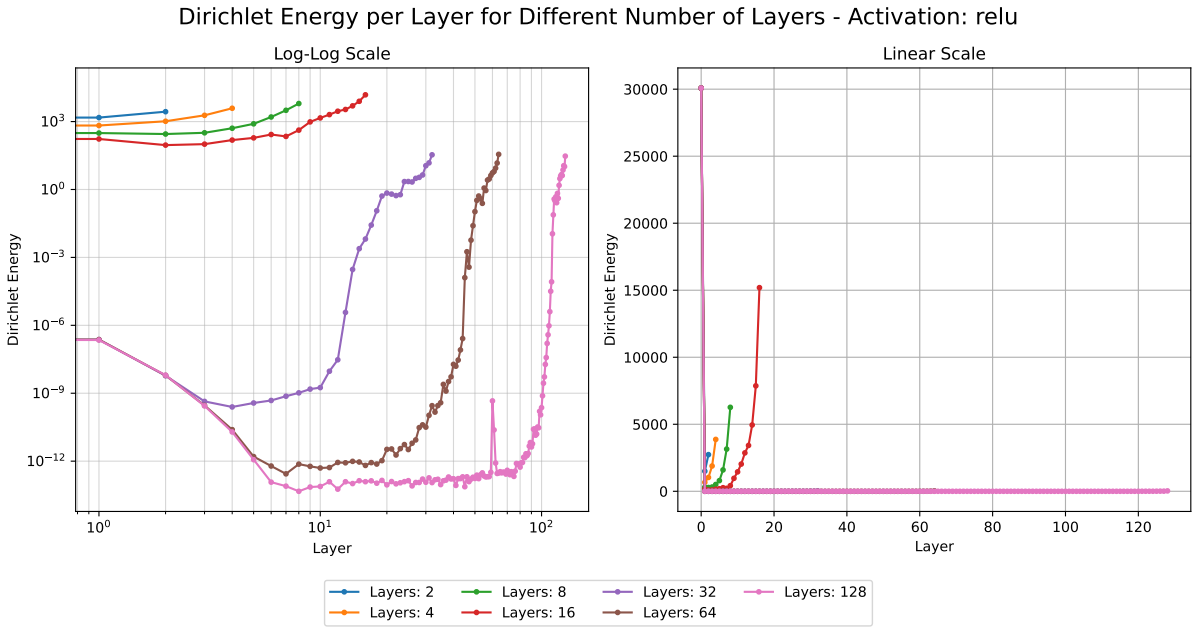

Figure 8: Dirichlet Energy per Layer for Different Number of Layers - ReLU Activation

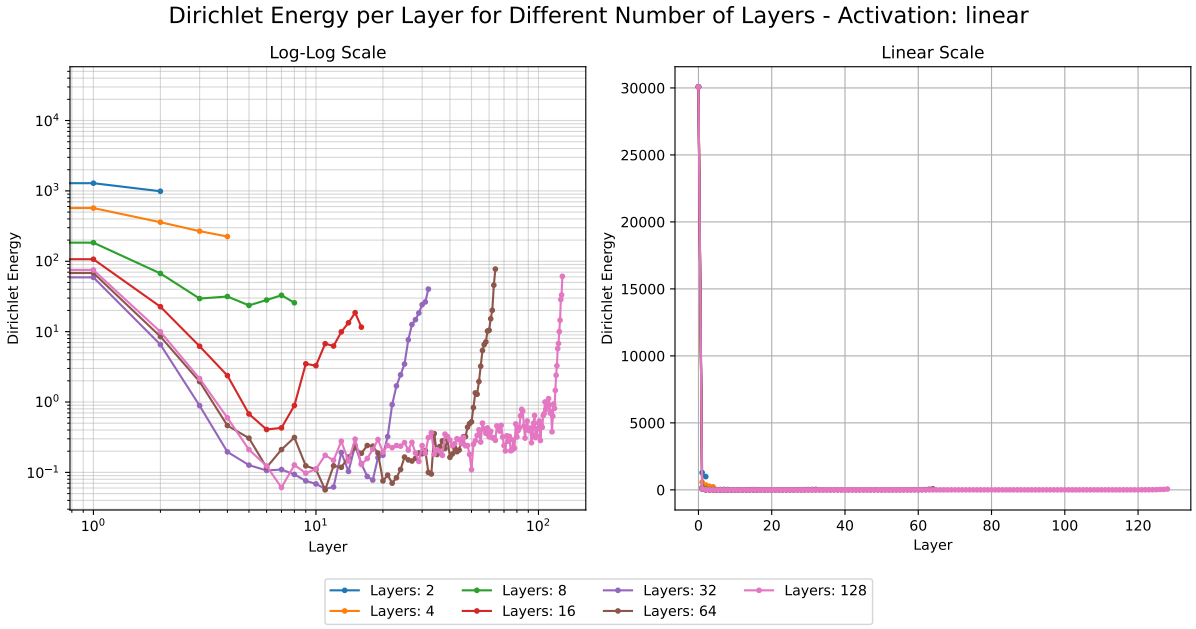

Figure 9: Dirichlet Energy per Layer for Different Number of Layers - Linear Activation

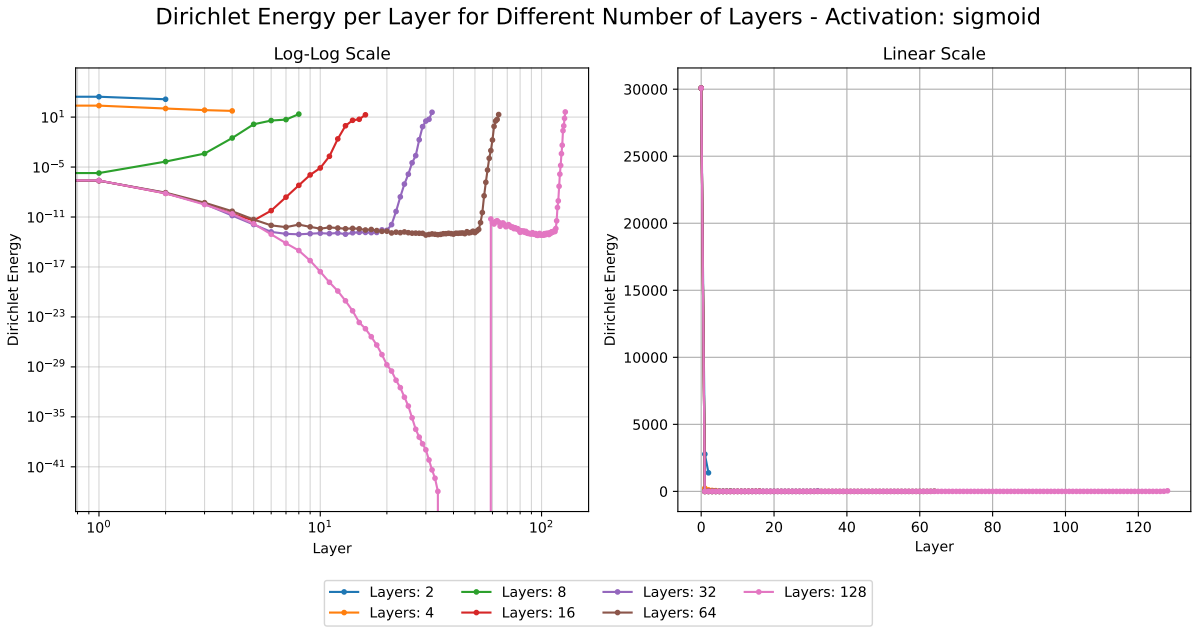

Figure 10: Dirichlet Energy per Layer for Different Number of Layers - Sigmoid Activation

Figures 8, 9 and 10 show the Dirichlet energy per layer for all the different layers of every tested model for the different activation functions.

We find that the Dirichlet energy is immediately the lowest after the first layer for the Sigmoid activation function. This is due to the squashing effect of the sigmoid function, which reduces the variance in the node features. The Dirichlet energy then decreases further after every layer, before increasing again in the last layers due to the use of Cross-Entropy Loss. For the 128 layer model, we observe the Dirichlet energy becoming so small, that the computer can no longer distinguish it from zero. This indicates that the node features have become nearly identical. Overall figure 10 shows an oversmoothing effect in addition to the squashing effect.

For the Linear activation function, we observe the highest Dirichlet energy after the first layer in all models. This is because the linear activation function does not modify the node features at all, preserving their variance. However, we still observe a strong decay in Dirichlet energy for deeper models, indicating that oversmoothing is still present. Yet, because the linear activation function does neither squash (sigmoid) nor threshold (ReLU) the node features, the energy stays higher for longer.

For the ReLU activation function, we again observe a decay in Dirichlet energy for deeper models, indicating the presence of oversmoothing. The ReLU activation function thresholds negative values to zero, which reduces the variance in the node features compared to the linear activation function. However, it does not squash large positive values like the sigmoid function, allowing it to maintain a higher Dirichlet energy than the sigmoid activation.

References

[1] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks, 2023.