Report HW3

Ruben Sasson: 342637721
Raphael Cohen: 345237721

Introduction

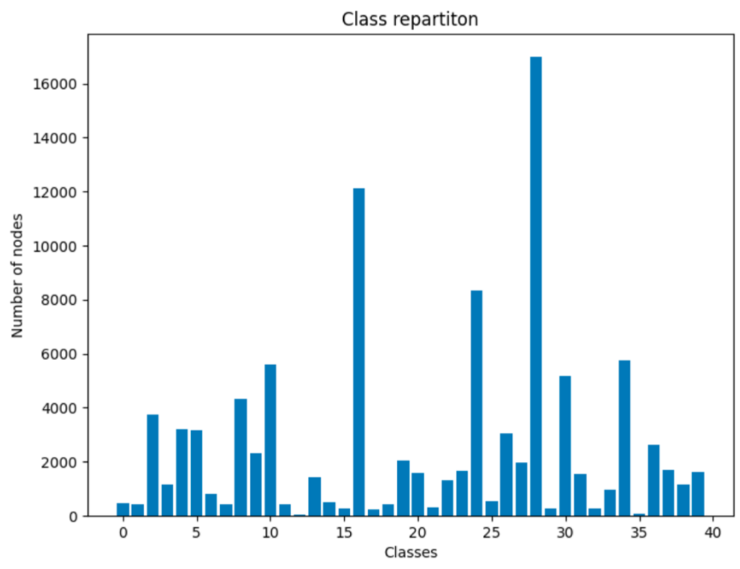
The primary objective of this assignment is to develop a sophisticated model capable of effectively classifying academic research documents, commonly known as papers, within the context of a university setting. Each paper is regarded as a node within a comprehensive graph structure, where the connections, or edges, between any two papers represent references from one paper to another. Furthermore, each paper is enriched with a multitude of features, totaling 128 in number, encompassing vital information such as the year of publication and an abstract embedding.
This report embarks on an insightful journey to analyze the performance of three distinct Graph Neural Network models. The focus lies on discerning the model that demonstrates superior classification capabilities when applied to this specific task. The ultimate objective is to achieve the highest level of accuracy possible when dealing with this meticulously curated dataset.
Moreover, this report delves into the intricacies of the dataset itself, unraveling the preprocessing steps undertaken and presenting a comprehensive analysis of its underlying characteristics. By shedding light on these aspects, a solid foundation is established for comprehending the subsequent discussions on the models employed, the crucial hyperparameters at play, and the resulting performance metrics.

Dataset analysis

First, our dataset comprises a total of 100,000 nodes, with each node representing a distinct academic research paper. Notably, every paper within the dataset possesses an embedding and is associated with a specific class, ensuring the absence of any missing data. The dataset is partitioned into an 80% training set and a 20% test set to facilitate model evaluation.

Upon examining the distribution of classes within the dataset, we observe that it is imbalanced. Certain classes are significantly underrepresented, while others, such as classes 28 and 16, are substantially overrepresented. Addressing this class imbalance poses a unique challenge, particularly since under sampling techniques would disrupt the inherent structure of the graph and its interconnectedness. As a result, we have opted to assign different weights to each class within the loss function, a topic that will be further discussed in the model section.

Additionally, it is crucial to highlight that certain classes, notably classes 12 and 35, exhibit minimal representation within the dataset. This poses a significant challenge during the prediction phase of our model.



Class repartition of the dataset

We have also noticed that the average number of connections per node is 8, but there are certain nodes that are particularly central in the graph (such as nodes 13573, 723, or 36094, which have been referenced over 3000 times). This high variance in connections makes the graph very challenging to partition, necessitating the treatment of the data as a single block rather than sub-blocks (batches). As a result, we will need to perform backward propagation after predicting the entire training set rather than after each batch. This approach requires a significant amount of memory on our computer.

Model

We tested two different models for this assignment. Both models are Graph Neural Network models that take as input the features of each node as well as the entire graph structure and its connections. Therefore, we were able to compare the results of a Graph Convolutional Neural Network (GCN) model and a Graph Attention Neural Network (GAT) model.
The GCN model utilizes graph convolution layers to aggregate information from neighboring nodes and focuses on local information exchange among the nodes in the graph. On the other hand, the GAT model uses attention mechanisms to assign weights to the connections between nodes, allowing for a more fine-grained focus on the most important relationships. By comparing the results of the two models, we were able to evaluate their respective classification performance accuracy. Additionally, we made adjustments to the hyperparameters of each model to optimize their performance.
In the next section, we will provide a more detailed explanation of the architectures of these models, as well as the specific choices we made for the hyperparameters. We will also analyze the obtained results and discuss the conclusions that can be drawn to select the most effective model for this research paper classification task. Our model is node

Graph Convolutional Neural Network

A Graph Convolutional Neural Network (GCN) is a type of neural network specifically designed to work with graph-structured data. It extends traditional convolutional neural networks (CNNs) to operate on graph data by leveraging the connectivity information inherent in the graph structure.
GCNs perform node-level feature aggregation by considering the features of a node's neighbors and updating the node's representation accordingly. This is achieved by applying a graph convolution operation, which combines the features of a node with the features of its neighbors, weighted by the graph connectivity. The goal is to capture both local and global information from the graph.
In the context of multi-class classification, a GCN can be used to classify nodes (in this case, research papers) into multiple classes. The final layer of the GCN model is typically a fully connected layer followed by a softmax activation function. This allows the model to produce a probability distribution over the classes for each node, indicating the likelihood of the node belonging to each class.

Architecture of our GCN model

- The input to the model consists of features for each node in the graph, with a total of 128 features (NUM_FEATURES).
- The hidden dimension of the model is set to 64 (HIDDEN_DIM).
- The model comprises two graph convolutional layers (GCNConv). The first layer takes the input features and performs the initial graph convolution, transforming the input features to a hidden representation.
- The hidden representation is then passed through a rectified linear unit (ReLU) activation function to introduce non-linearity.
- Dropout is applied with a probability of 0.5 during training to prevent overfitting.
- The second graph convolutional layer is applied to the updated hidden representation, further refining the node features.
- Another ReLU activation and dropout are applied to the output of the second layer.
- Finally, the output is passed through a fully connected layer (nn.Linear) to map the hidden representation to the number of classes (NUM_CLASSES).
- The softmax activation function is applied to produce class probabilities, indicating the likelihood of each class for each node.

Graph Attention Neural Network

A Graph Attention Neural Network (GAT) is a type of neural network designed for graph-structured data that utilizes attention mechanisms to capture important relationships between nodes. It extends the concept of self-attention from Transformer models to operate on graph data.
GATs compute attention coefficients for each edge in the graph, allowing the model to assign different weights to the contributions of neighboring nodes. This enables the network to focus on the most relevant nodes during message passing and feature aggregation. The attention coefficients are determined by a learnable attention mechanism that takes into account the node features and their connectivity.
In the context of multi-class classification, a GAT can be used to classify nodes (research papers in this case) into multiple classes. Similar to GCNs, the final layer of the GAT model is

typically a fully connected layer followed by a softmax activation function, producing class probabilities for each node.
Compared to Graph Convolutional Neural Networks (GCNs), GATs have the advantage of incorporating a more fine-grained attention mechanism. GCNs aggregate information from neighboring nodes uniformly, whereas GATs assign different weights to neighbor nodes based on their importance. This allows GATs to capture more nuanced relationships and prioritize relevant information during message passing.
Architecture of our GAT model

- The input to the model consists of features for each node in the graph, with a total of 128 features (NUM_FEATURES).
- The hidden dimension of the model is set to 64 (HIDDEN_DIM), and the number of attention heads is set to 10 (HEADS).
- The model comprises two GAT layers (GATv2Conv). The first GAT layer takes the input features and computes attention coefficients to aggregate information from neighboring nodes.
- An exponential linear unit (ELU) activation function is applied to introduce non-linearity.
- Dropout is applied with a probability of 0.6 during training to prevent overfitting.
- The second GAT layer is applied to further refine the node representations by considering the attention-guided neighbor information.
- Another ELU activation and dropout are applied to the output of the second GAT layer.
- The hidden representation is then passed through two linear layers (nn.Linear) to transform the features to the desired output dimensionality.
- ELU activations are applied after each linear layer to introduce non-linearity.
- Finally, the output is passed through a fully connected layer (nn.Linear) to map the hidden representation to the number of classes (NUM_CLASSES).
- The logarithmic softmax activation function (F.log_softmax) is applied to produce log-probabilities of each class for each node.

Training and evaluation

For the GCN Model:

we trained it with the following parameters: a learning rate of 0.01, 2000 epochs, and a hidden dimension of 64. The model training was relatively fast, taking approximately 1 hour to obtain results. It's important to note that we did not divide the training set into batches since we provided the entire graph as input for each iteration. Therefore, we fed the entire training set before performing backward propagation.
During training, we evaluated the model on the test set. We utilized the Cross Entropy loss function and experimented with two different types of loss. The first type assigned equal weights to all errors, regardless of the class. The second type assigned different weights to errors based on the class where the error occurred. The weight assigned to an error was greater for classes that were underrepresented and vice versa. This technique helped to counterbalance the class imbalance within our dataset.

For the GAT model:

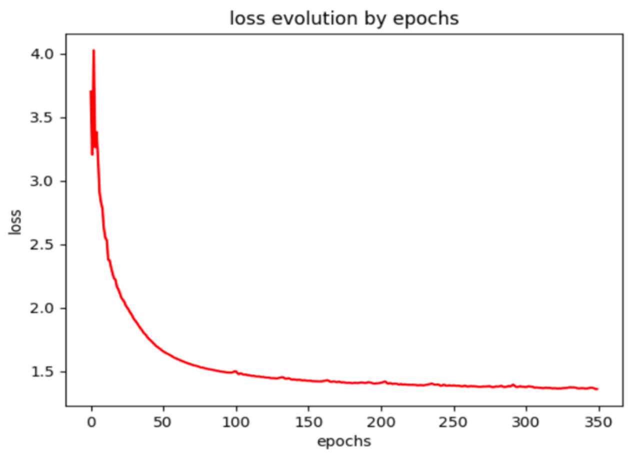
we trained it with 12 attention heads, a learning rate of 0.005, and 350 epochs. Training the GAT model took significantly longer due to the increased number of layers and the computational overhead of the attention mechanism.
Similar to the GCN model, we employed the Cross Entropy loss function for training the GAT model. We tested the model twice: once with uniform weights for all classes and another time with weights varying based on the class representation within the dataset. The varying weights approach aimed to address the class imbalance by assigning higher weights to underrepresented classes.

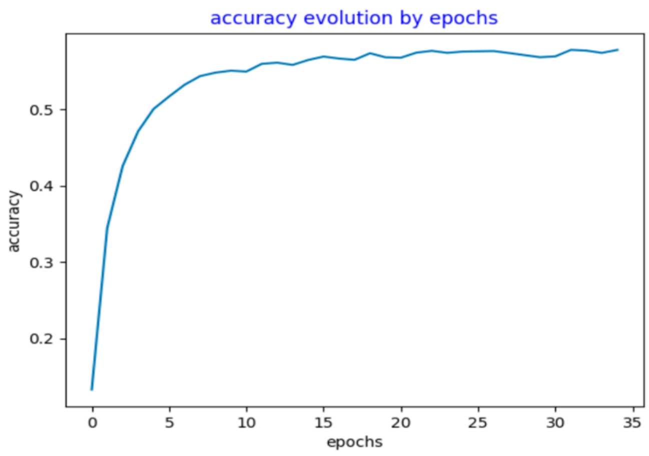
Results:

Model	**GCN with non - weighted loss**	**GCN with non - weighted loss**	**GAT with non-weighted loss**	**GAT with weighted loss**
Accuracy	0.35	0.13	0.58	0.51
Loss	3.11	3.77	1.23	1.52

After running all the models, we achieved an accuracy of 0.58 with the Graph Attention Neural Network model using uniform weighting for the loss function. This result is quite good considering the imbalanced nature of the data. Our model demonstrates an overall ability to correctly classify the categories that are well-represented in the dataset. However, none of the models were able to accurately classify the categories that are underrepresented.

We have included figures depicting the evolution of the loss and accuracy over the epochs for the selected model. These figures provide insights into the model's training process and the performance improvements achieved over time.





Conclusion

In this homework, we tested two types of models, each with two types of loss functions, and proposed different architectures for each model. The Graph Convolutional Neural Network (GCN) initially did not yield satisfactory results, as it struggled to converge the loss towards an acceptable minimum. Furthermore, using a weighted loss function based on class representation in the dataset did not yield favorable results for either model.
On the other hand, the utilization of a Graph Attention Neural Network (GAT) proved to be more successful. We achieved an accuracy of 0.58 without employing any weighting in the loss function. This indicates that the GAT model performed better in handling the challenges posed by the imbalanced dataset.
Several hypotheses can be explored to further improve the model. One approach would be to enrich the dataset and aim for better class balance, ensuring more even representation across classes. Additionally, increasing the number of attention heads or hidden layers could potentially improve the model's performance. Another option would be to group together similar classes to reduce the number of distinct classes, thereby simplifying the prediction task.
Overall, our findings highlight the effectiveness of the Graph Attention Neural Network in tackling the task of classifying academic research papers. However, there is room for further experimentation and exploration of different strategies to enhance the model's performance and address the challenges associated with imbalanced data.

Submitted files

- **Dataset.py** : the file that allows to load the data
- **Analyse.py** : all the functions that allows to analyze and visualize the dataset. Including statistics and class repartition
- **GAT_classification.py** : Contains the model GAT, the hyper parameter of the model, the training parameters, the training function and the evaluation function
- **GNN_classification.py** : Contains the model GCN, the hyper parameter of the model, the training parameters, the training function and the evaluation function
- **Utils.py :** Contains some utils function such as the function for mask creation, subgraph visualization and the function of weight-class attribution for the weighted loss
- **Predict.py:** The file that load our model and test it to the evaluation set.

Link to the GitHub repository : https://github.com/Raphico2/HW3_lab_Technion/tree/main