DSC 180A Capstone Quarter 1 Project Report - Section B12

Gita Anand
giaand@ucsd.edu

Gal Mishne
gmishne@ucsd.edu

Yusu Wang
yuw122@ucsd.edu

Abstract

Graph Neural Networks are a widely-used data structure in the machine learning world today because of their task versatility and their ability to represent relationships between individual data points, effectively reducing overfitting. One use case for graph neural networks is geometric deep learning, where neural networks can also learn from non-euclidian data. However, while GNNs are highly effective in node prediction, link prediction, learning over an entire network, and community detection, they still vary in their effectiveness due to phenomena such as "oversmoothing" and "oversquashing", which both detrimentally affect the functional accuracy of GNN machine learning tasks. In this paper, we aim to investigate the implementation and performance of various GNN models, including GIN, GCN, GATv2, and GraphGPS for node and graph classification tasks, as appropriate, across multiple datasets obtained from the Python PyTorch Geometric package. We will empirically validate performance metrics of these graph architectures, and form conclusions regarding the use cases of our models given our evaluation.

Code: https://github.com/ganand01/DSC180A-Quarter1Project.git

1 Introduction

Graph neural networks are a machine learning data structure, containing nodes, or entities that hold information, and edges, or connections between nodes that contain associative pieces of information. A graph is very flexible in that it could be viewed as a way of structuring data, or could be used as a single datapoint, linked to several other graphs to form a network. The versatility of these data structures are often taken advantage of for deep learning purposes. This is a major breakthrough, because the relationships can also be included in the implementation of classification, prediction, and other algorithms.

The concept of Geometric Deep Learning, which aims to build neural networks that can learn from non-euclidian while also exploiting the inherent geometric structure of the data, further extends the utility of GNNs. For example, the act of predicting the labels of people posing in three-dimensional space instead of two-dimensional space, meaning that every pixel represents a point cloud that wraps around the 3-D object, would be considered geometric deep learning. While traditionally associated with non-graph data, geometric deep learning is also highly effective with graph-based data. By recognizing and leveraging feature relationships, it mitigates the "curse of dimensionality" – a challenge in high-dimensional data spaces.

While geometric deep learning is advantageous in that it can assist in combating the curse of dimensionality by finding relationships through the data, it also has certain drawbacks when applied to graph neural network (GNN) models. Some of these drawbacks are known as "oversmoothing" and "oversquashing". The oversquashing phenomenon happens when particular nodes have higher weights than others, causing the model to have emphasis on certain nodes, while deeming other, equally important nodes as irrelevant. This increases the weight of some features in the model at the cost of other features, and results in a loss of attention to detail. In contrast, oversmoothing occurs in graph neural networks as a result of a large depth in the network. When the number of layers increases, the overload of features when aggregated makes neighboring nodes more similar, oversimplifying the data structure. Eventually, the majority of nodes will have similar feature representation, so that the nodes are no longer unique.

To understand and improve the representational power of GNNs, it's crucial to consider their ability to distinguish between different graph structures. One mainstream technique to broadly classify these architectures through similarity is the Weisfeiler-Lehman (WL) graph isomorphism test, a method that iteratively updates node features by aggregating features from neighboring nodes. The strength of the WL test lies in its injective update mechanism, which uniquely maps node neighborhoods to distinct feature vectors. Similarly, a GNN can match the discriminative power of the WL test if it employs an expressive and injective aggregation scheme.

In this paper, we aim to investigate the performance and applicability of various GNN models, including GIN, GCN, GATv2, and GraphGPS, in node and graph classification tasks. Using datasets from the PyTorch Geometric library, particularly the Long-Range Benchmark Dataset Pascal, the IMDB dataset, the Cora dataset, and the Enzymes dataset, we will empirically validate the effectiveness of these architectures, exploring their strengths and

limitations in graph-based learning.

2 Methods

2.1 Introduction to Datasets

`IMDB-BINARY`: The IMDB-BINARY dataset consists of ego-networks of 1,000 actors and actresses who have appeared on movies that are in the IMDB database in primarily the Action and Romance genres. This is fitting given that ego networks are typically used for analyzing social connections and relationships. In each graph, the nodes represent actors, while the edge is binary, based on whether they have ever appeared in the same movie or not.

`PASCAL VOC`: The PASCAL VOC, or Pascal Visual Object Classes dataset is a long-range benchmark dataset used for object detection, segmentation, and classification, widely used for computer vision. The dataset contains 20 object object categories, and features 1464 images for training, 1449 images for validation, and a private testing dataset.

`ENZYMES`: The Enzymes dataset was derived from the BRENDA database (Schomburg et al., 2004). The machine learning task associated with this dataset is to assign enzymes to one of the 6 EC top-level classes, which reflect the catalyzed chemical reaction.

`CORA`: The Cora dataset is a popular benchmark dataset containing 2708 scientific publications, which are classified into 7 distinct classes, with 1433 unique words. In this instance, each publication is a node of the graph, while each citation is considered an edge of the graph, and the unique words are input as the binary node features of the graph, indicating the presence or absence of a word in the publication. This dataset is widely used to predict the class of each publication, or node in the graph.

2.2 Message Passing Network (MPN)

A message passing network, or an MPN, is designed to process graph-structured data, and perpetuate the processing and exchanging of information between nodes of a graph, which, in turn, allows the network to learn the representation of every node. The intricacies of this process can be outlined in simplistic terms. First, a message function will compute messages based on neighboring node features. Particularly, a message passed from neighbor node x to node y at the kth iteration, where N is the set of neighbors can be denoted as $m_{uv}^{(k)} = f(h_u^{(k-1)}, h_v^{(k-1)})$. Then, all of node y's messages are aggregated through some function before being combined with its previous feature vector using some differentiable function. The choice of aggregation function in GNN, however, is a crucial part of the graph architecture development process. This process is performed iteratively to keep each node up to date on information being received from neighboring nodes.

Now, we will move onto the use cases of GNN models for the tasks of interest to us in this

research paper. The first is node classification, a task where each node v has an associated label y, and the deep learning task is to learn a representation vector, so that the labels of node v can be predicted. The second is graph classification, a task where a set of graphs is the input along with their associated labels, and a representation vector is learned that helps predict the graph-wide labels. Both of these tasks use the technique of back propagation for training of the NNs.

2.3 Graph Isomorphism Network (GIN)

Out of all GNN models that have been publicly released and are being widely used and accepted in the deep learning community, the GIN, or Graph Isomorphism Network is considered the most efficient and accurate. This is because compared to GCNs (Graph Convolutional Networks) and GAT (Graph Attention Networks), GINs are designed to be optimal in terms of capturing the properties of graphs. Due to the Weisfeiler-Lehman (WL) graph isomorphism test which is unique to the GIN implementation, GINs can better distinguish between different graph structures when compared to other models, which use aggregate neighbor information, which dilutes the accuracy metric. This means that GINs are currently the most powerful in learning representations that differentiate between graphs that are not isomorphic, and are less prone to phenomena such as oversmoothing and oversquashing.Finally, GIN does not include normalization in its aggregation function, which also has been observed to increase overall test accuracy.

Now, let us delve into the mathematical intricacies of GIN, and how the graph is structured to optimize accuracy. The GIN architecture foundationally based upon the WL test as a heuristic, which refines node labels until the point of stabilization. In each layer of the GIN architecture, a node's features and its neighboring features are updated based on the following formula: $h_v^{(k)} = MLP^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right)$, where h is a feature vector for node v at the kth layer, N is the set of neighboring nodes of v, and MLP refers to the multi-layer perceptron at the kth layer. This equation is a key component of the GIN architecture because it enables complex transformation over the data that has been aggregated over the immediate neighborhood, and adjusts weight of the node and its neighboring node features.

2.4 Graph Attention Networks (GAT)

The Graph Attention Network (GAT) is another GNN architecture widely used for node and graph classification. This architecture is particularly unique in its use of attention mechanisms, which gives additional weight to sections of the graph that are deemed more relevant for a specific task. This is done through dynamic weight assignment, where the weights will change according to the task being performed, and certain nodes and edges will have more influence on the output of the graph.

This particular graph architecture starts by transforming each node feature using a linear transformation, before then applying the aforementioned attention mechanism. In the

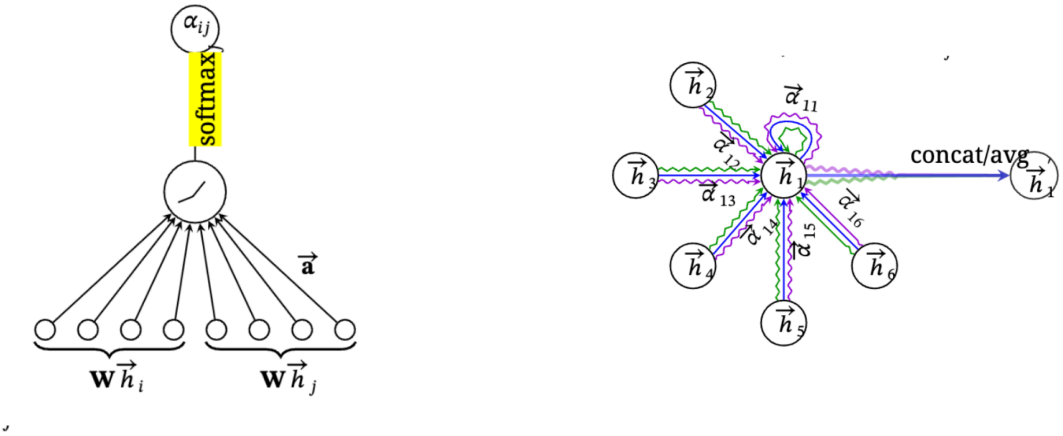

Figure 1: GAT Architecture Diagram

context of a GAT, the attention mechanism works to compute the attention coefficient of each node-pair, and weight the priority of each of the node features. In this step, masked attention is performed, where attention coefficients are only computed for notes in the neighborhood of a particular node within the graph, and all attention coefficients are normalized using the softmax function, $\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$, which makes it easier to differentiate the more relevant features of a node's neighbors. These attention coefficients are then aggregated, and then the activation function LeakyReLU is applied. Finally, the attention mechanisms are independently concatenated into one feature representation output, a step known as "multi-headed attention".

2.5 General Powerful Scalable Graph Transformer (GraphGPS)

The GraphGPS Transformer model, or the General Powerful Scalable Graph Transformer, is a new graph neural network architecture, published in January 2023 that is designed for scalable processing of graph data. It integrates features such as positional encodings into the Message Passing Neural Network (MPNN) architecture, which takes advantage of the nodes position within space and computes pairwise distance, and structural encoding, which provides an embedding of the graph structure, and helps with generalizability of the graph. The use of positional encodings (PE) and structural encodings (SE) is performed using the features at the node, edge, and graph levels respectively.

It also introduces the idea of a GPS layer, which is a hybrid MPNN and transformer layer, where the features are updated by aggregating the MPNN output with the global attention layer, which, as mentioned before, gives weighted relevance to certain nodes and edges over others. This aggregation step allows for the capturing of long-range dependencies across the entire graph, and helps with understanding of the overall graph structure, regardless of the size of the graph.

3 Results

In our comparative study of graph neural network models, we implemented and trained four different architectures: Graph Convolutional Network (GCN), Graph Isomorphism Network (GIN), Graph Attention Network v2 (GATv2), and a graph transformer model, GraphGPS. These models were evaluated on the Cora, IMDB, Enzyme datasets, and a dataset from the Long Range Graph Benchmark (PASCAL-VOC) to gain comprehensive insights into performance on graph-structured data.

Each model was constructed with specific implementation details to optimize performance. The architectures were designed with multiple layers, where the GIN and GATv2 models comprised three convolutional layers. I also made sure to experiment with different numbers of layers in my deep learning network, observing which models performed better and which ones performed worse. A consistent approach to optimization was employed, utilizing the Adam optimizer with learning rates tailored to each model to balance convergence speed and stability. The learning rates varied from 0.005 to 0.01, depending on the dataset and model complexity. The models were trained over 100 epochs on the majority of architectures, excluding GraphGPS, which is lengthy in its runtime. This was done to ensure sufficient learning while avoiding overfitting.

Below are the results after running all my code. As you can see, each model has extremely varied accuracies, and while some are caused by inefficient implementation of pre-existing models,and different levels of model optimization, the inherent efficiency of the model should be factored into its accuracy.

Table 1: Test Accuracies of Graph Neural Network Architectures

Architecture	Cora	IMDB	Enzymes	Pascal
GIN	0.662	0.797	0.6953	0.6953
GCN	0.759	0.822	0.4033	0.4033
GATv2	0.782	0.69	0.2367	0.6953
GraphGPS	0.551	0.753	0.0	—

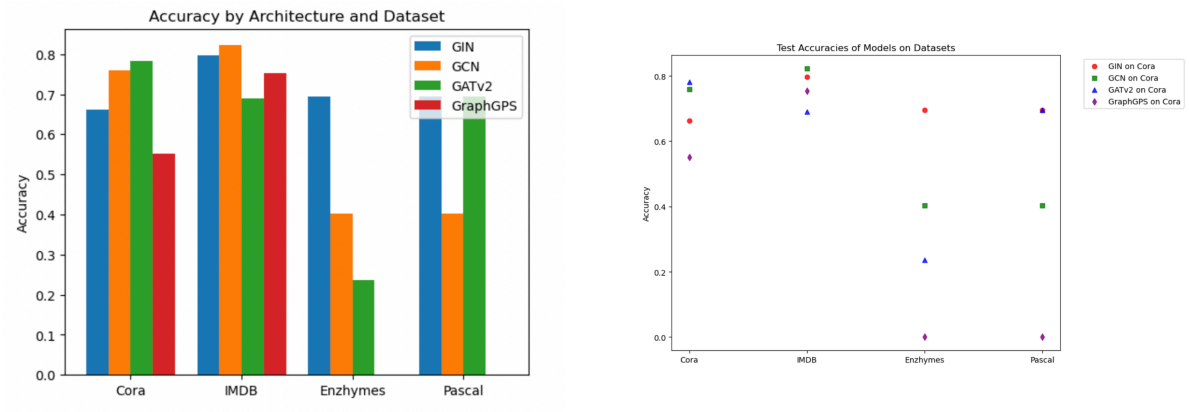

Figure 2: Scatter Plot and Bar Graph of GNN Architecture Accuracies

4 Conclusion

The exploration of various Graph Neural Network (GNN) architectures in this study has provided valuable insights into their performance across different types of graph-structured data. The Graph Convolutional Network (GCN), Graph Isomorphism Network (GIN), Graph Attention Network v2 (GATv2), and GraphGPS models each have their strengths and weaknesses, as demonstrated by their performance on the Cora, IMDB, Enzyme, and PASCAL-VOC datasets.

Our findings suggest that the choice of architecture is crucial and should be tailored to the specific characteristics of the dataset. For instance, GCN and GIN showed promising results on the Cora and IMDB datasets, which can be attributed to their ability to capture local neighborhood structures effectively. On the other hand, the GATv2's attention mechanism provided it with the flexibility to focus on relevant parts of the graph, which proved beneficial for certain datasets.

It is also evident that the tuning of hyperparameters, such as the number of layers, learning rate, and the number of training epochs, plays a significant role in model performance. The choice of optimization algorithm and the careful balancing of convergence speed against the risk of overfitting are critical to achieving high accuracy.

GraphGPS's transformer-based approach indicated the potential for handling more complex graph structures, although it requires careful tuning and sufficient computational resources due to its longer runtime. The GPU required to run GraphGPS effectively caused us problems in the implementation, and we were unable to implement it efficiently, but it shows scope for being a state-of-the-art architecture.

In conclusion, this study underscores the importance of understanding both the dataset at hand and the GNN architecture being used. Future work may involve deeper exploration into hyperparameter optimization, the use of more sophisticated regularization techniques to prevent overfitting, and investigating the scalability of these models to larger and more complex graphs. Furthermore, the implementation of advanced techniques such as trans-

fer learning and unsupervised pre-training could also be explored to enhance the models' performance.

By continuing to refine these GNN models and their implementations, we can better harness the power of graph-structured data across a myriad of applications, from social network analysis to bioinformatics and beyond.

References

[1] P. Veličković, et al. "Graph Attention Networks." *arXiv preprint arXiv:1710.10903*, 2017. Available: https://arxiv.org/abs/1710.10903

[2] W. Hamilton, et al. "Representation Learning on Graphs: Methods and Applications." *arXiv preprint arXiv:1810.00826*, 2018. Available: https://arxiv.org/abs/1810.00826

[3] T. Kipf. "Graph Convolutional Networks." Available: https://tkipf.github.io/graph-convolutional-networks/

[4] The Modern Scientist. "Graph Neural Networks Series Part 4: The GNN's Message Passing & Over-Smoothing." Available: https://medium.com/the-modern-scientist/graph-neural-networks-series-part-4-the-gnns-message-passing-over-smoothing-e77ffee523cc

[5] F. Tong. "What is Geometric Deep Learning?" Available: https://flawnsontong.medium.com/what-is-geometric-deep-learning-b2adb662d91d

[6] Towards Data Science. "Machine Learning Tasks on Graphs." Available: https://towardsdatascience.com/machine-learning-tasks-on-graphs-7bc8f175119a

[7] Towards Data Science. "Graph Theory and Deep Learning Know-Hows." Available: https://towardsdatascience.com/graph-theory-and-deep-learning-know-hows-6556b0e9891b

[8] F. Wu, et al. "Self-Supervised Graph Transformer on Large-Scale Molecular Data." *arXiv preprint arXiv:2007.08663*, 2020. Available: https://arxiv.org/abs/2007.08663

[9] V. P. Dwivedi, et al. "Recipe for a General, Powerful, Scalable Graph Transformer." *arXiv preprint arXiv:2205.12454*, 2022. Available: https://arxiv.org/abs/2205.12454