Analysis of Graph Neural Networks on Amazon Co-Purchase Graph

Final Project Report

Vincent Wilmet
vincent.wilmet@student-cs.fr
CentraleSupélec
Gif-sur-Yvette, France

Suer Lin
suer.lin@student-cs.fr
CentraleSupélec
Gif-sur-Yvette, France

Lu Wang
lu.wang@student-cs.fr
CentraleSupélec
Gif-sur-Yvette, France

Asrorbek Orzikulov
Asrorbek.Orzikulov@student-cs.fr
CentraleSupélec
Gif-sur-Yvette, France

ABSTRACT

The multi-class classification problem has been there for decades, and many algorithms have been developed to deal with this task. Formerly, the data used for a classification task was in a tabular form. However, in the last 10 years, various attempts were done to develop and test classification models that work on graph data. In this study, we compared 5 such graph neural networks on a very challenging dataset, the Amazon product co-purchasing data. To make the task more realistic, we utilized a sales-based split, using more popular products as training examples and less frequent ones as a test set. Our experiments show that adaptive Graph Attention Networks perform the best and have the narrowest gap between the training accuracy and the test accuracy. Meanwhile, models using simple Graph Convolutional Layers had the lowest accuracy and the widest generalization gap.

INTRODUCTION

Motivation

Many classification algorithms have been developed to reveal the properties of complex networks. However, how good the algorithm is in terms of accuracy and computation time is still open. Therefore,in this study,we use the ogbn-products dataset from the Stanford Open Graph Benchmark (OGB) [6] to compare five different algorithms based on accuracy.

Problem definition

Relationships between components of complex systems can be represented by networks. In this description, the elements that make up the system are described as nodes, and their interactions are described as edges.

In this project, we did experimental evaluation of algorithms on co-purchasing graph data from the Amazon product co-purchasing dataset. Our goal is to correctly predict the category of a product in a multi-class classification setting, where the top-level categories are used for target labels, making it a node classification task. So, we aim to maximize the likelihood of observing the target categories at hand. This essentially means minimizing the negative log-likelihood, which is defined as follows:

$$J_{NLL} = -\frac{1}{N}\sum_{x\in X}\sum_{c=1}^{C} y_c \log\left(\hat{y}_c\right) \tag{1}$$

where X are the features, y_c are ground-truth values and $\hat{y}_c$ are predicted probabilities. Since all our models output $\log\left(\hat{y}_c\right)$, we used this lost function.

For evaluation purposes, model accuracy is used. Accuracy is the degree to which predictions match true values.

Finally, instead of randomly splitting the dataset into 90% as the training set and 10% as the test set, we denoted the top 8% as the training set, the next 2% as the validation dataset, and the remaining 90% as the test set based on the sales rank. This splitting process made the prediction more challenging, but it is closer to practical applications. In real life, it is likely that more frequent products are labeled first and used in training, and then predictions are done for a lot of less frequent products.

In practice, we download the data from the ogn library via the following lines of code:

```
from ogb.nodeproppred import PygNodePropPredDataset
fp = '/content/drive/MyDrive/MLNS/final_project/'
dataset = PygNodePropPredDataset('ogbn-products', fp)
```

It is 1.3GB in size. For more details, visit the **Exploratory Data Analysis** section.

RELATED WORK FOR THE PROBLEM

In this paper, we outline a few models that would solve the problem previously defined above.

However, it is important to review from the beginning. Early methods based on recursive neural networks focus on dealing with directed acyclic graphs [13]. Later, the concept of graph neural network (GNN) [12] was first proposed in 2009 by Scarselli et al., which extended existing neural networks to process more graph types. The target of a GNN is to learn a state embedding $h_v \in \mathbb{R}^s$ which contains the information of the neighborhood and itself for each node. It can directly process most of the practically useful types of graphs, e.g., acyclic, cyclic, directed, and undirected. Our models, and other modern models are built on the foundational work of these researchers.

Rather than using a recurrent operator for the propagation step, many modern approaches use a convolution operator. We looked at two main approaches to the convolution operator:

1) **Spectral Approach**: These methods are theoretically based on graph signal processing and define the convolution operator in the spectral domain [8]. Based on the convolution theorem, the convolution operation is defined as:

$$\mathbf{g}\bigstar\mathbf{x} = \mathfrak{I}^{-1}(\mathfrak{I}(\mathbf{g}) \odot \mathfrak{I}(\mathbf{x})) \tag{2}$$

$$= \mathbf{U}(\mathbf{U}^T\mathbf{g} \odot \mathbf{U}^T\mathbf{x}) \tag{3}$$

where $\mathbf{U}^T\mathbf{g}$ is the filter in the spectral domain, which can be simplified using a learnable diagonal matrix $\mathbf{g}_w$ which forms the basic function of spectral methods:

$$\mathbf{g}_w\bigstar\mathbf{x} = \mathbf{U}\mathbf{g}_w\mathbf{U}^T\mathbf{x} \tag{4}$$

Our first model type, and baseline graph network draws on the aforementioned spectral approach by updating the spectral method defined in (4). In 2017, the authors Kipf & Welling created the Graph Convolutional Network (GCN) [7] by simplifying the convolution operation in (4) alleviate the problem of overfitting and the exploding vanishing gradient problem in (7).

$$\mathbf{g}_w\bigstar\mathbf{x} \approx \Sigma_{k=0}^{K} w_k\mathbf{T}_k(\hat{(L)})\mathbf{x} \tag{5}$$

$$\mathbf{g}_w\bigstar\mathbf{x} \approx w_0\mathbf{x} + w_1(\mathbf{L} - \mathbf{I}_N)\mathbf{x} = w_0\mathbf{x} - w_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x} \tag{6}$$

$$\mathbf{g}_w\bigstar\mathbf{x} \approx w(\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{x} \tag{7}$$

Finally, the compact form of a GCN convolutional step is defined as:

$$\mathbf{H} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}\mathbf{W} \tag{8}$$

where the range of the eigenvalues in $\hat{(L)} = \frac{2}{\lambda_{max}}\mathbf{L} - \mathbf{I}_N$ is [-1,1]. $\mathbf{w} \in \mathbb{R}^K$ is a vector of Chebyshev coefficients. Chebyshev polynomials are defined as $\mathbf{T}_k(\mathbf{x}) = 2\mathbf{x}\mathbf{T}_{k-1}\mathbf{x} - \mathbf{T}_{k-2}\mathbf{x}$ with $\mathbf{T}_0\mathbf{x} = 1$ and $\mathbf{T}_1\mathbf{x} = \mathbf{x}$. $\mathbf{X} \in \mathbb{R}^{NxF}$ is the input matrix, $\mathbf{W} \in \mathbb{R}^{FxF'}$ is the parameter matrix and $\mathbf{H} \in \mathbb{R}^{NxF'}$ is the convolved matrix. F and F' are the dimensions of the input and the output, respectively.

2) **Basic Spacial Approaches**: Our second model type draws from spatial approaches which define convolutions directly on the graph based on the graph topology. The major challenge of spatial approaches is defining the convolution operation with differently sized neighborhoods and maintaining the local invariance of CNNs. The GraphSAGE model (SAmple and aggreGatE) [5] generates embeddings by sampling and aggregating features from a node's local neighborhood:

$$\mathbf{h}_{Nv}^{t+1} = AGG_{t+1}(\mathbf{h}_u^t, \forall u \in N_v) \tag{9}$$

$$\mathbf{h}_{Nv}^{t+1} = \sigma(\mathbf{W}^{t+1} \cdot [\mathbf{h}_v^t || \mathbf{h}_{N_v}^{t+1}]) \tag{10}$$

Instead of using the full neighbor set, GraphSAGE uniformly samples a fixed-size set of neighbors to aggregate information. We will go more in depth on the aggregation function AGG in (14) and(15)

3) **Attention based Spacial Approaches**: Our third model type, the Graph Attention network (GAT) [14] incorporates the attention mechanism into the propagation step. It computes the hidden states of each node by attending to its neighbors, following a self-attention strategy. The hidden state of node v can be obtained by:

$$\mathbf{h}_v^{t+1} = \rho(\Sigma_{u\in N_v}\alpha_{vu}\mathbf{W}\mathbf{h}_u^t) \tag{11}$$

$$\alpha_{vu} = \frac{exp(LeakyReLU(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v||\mathbf{W}\mathbf{h}_u]))}{\Sigma_{u\in N_v} exp(LeakyReLU(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v||\mathbf{W}\mathbf{h}_k]} \tag{12}$$

where $\mathbf{W}$ is the weight matrix associated with the linear transformation which is applied to each node, and $\mathbf{a}$ is the weight vector of a single-layer Multi-Layer Perceptron (MLP) [11]. The attention architecture has several properties: (1) the computation of the node-neighbor pairs is parallelizable thus the operation is efficient; (2) it can be applied to graph nodes with different degrees by specifying arbitrary weights to neighbors; (3) it can be applied to the inductive learning problems easily

Our algorithms are conceptually related to previous node embedding approaches, general supervised approaches to learning over graphs, and recent advancements in applying convolutional neural networks to graph-structured data. In this case, we opted for a graph neural network (GNN)-based approach to obtain an encoding function. This type of approach, although being initially proposed in the late 1990s and early 2000s [1] has been shown particularly effective for recommendation problems [10]. We also got inspiration that by borrowing information from nearby nodes/Pins the resulting embedding of a node becomes more accurate and more robust[2]. We draw on the 3 modern propagation step approaches to help us replicate 5 models. To see more in depth explanations of the models previous mentioned and what they do, see our section on **Models Used**.

METHODOLOGY

Models Used

We built 5 different models using graph convolutional layers that are popular in the academy and the industry. Each model contained only one type of layer, and attention-based models also had skip connections. Below, we presented a brief description for each of those layers.

GCN : Our baseline model includes the popular Graph Convolutional Layers [7]. For each node, we need to consider all its neighbors and the characteristic information they contain. Assuming that we use the function Average (), we can do this for each node and get an average representation that can be input into the neural network.

Each GCN layer has the following information propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{13}$$

where $\tilde{A} = A + I_N$ is the adjacency matrix with self-connections, σ is a non-linear activation function, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(l)}$ are trainable weight matrices. Since the equation is computationally expensive, it is estimated using a localized first-order approximation. This is the same equation as (8).

GraphSAGE : GraphSAGE (SAmple and aggreGatE) [5], uses SAGEConv and unlike embedding approaches that are based on matrix factorization, it leverages node features (e.g., text attributes, node profile information, node degrees) in order to learn an embedding function that can generalize to unseen nodes.

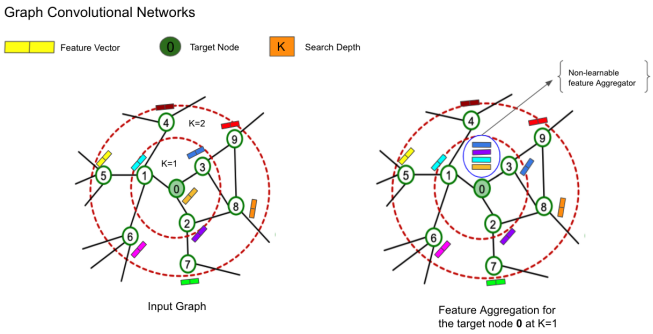

Figure 1: Graph Convolutional Networks [7]

It is developed on the basis of revolution and has the capability of generalization for unseen data. In the process of training, the neighbor nodes are sampled instead of training all nodes. The GraphSage algorithm exploits both the rich node features and the topological structure of each node's neighbourhood simultaneously to efficiently generate representations for new nodes without retraining.

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

1 $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
2 **for** $k = 1...K$ **do**
3 **for** $v \in \mathcal{V}$ **do**
4 $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
5 $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
6 **end**
7 $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
8 **end**
9 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Figure 2: GraphSAGE Algorithm [5]

The aggregator functions the authors proposed to fit in line 4 and 5 of the algorithm would be one of three following:

1) **Mean Aggregator**: Nearly equivalent to the convolutional propagation rule used in a GCN [7]. But we can derive an inductive variant of the GCN [7] via the following formula:

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot MEAN((\mathbf{h}_v^{k-1}) \cup (\mathbf{h}_u^{k-1}, \forall u \in N(v)))) \quad (14)$$

2) **Pooling Aggregator**: In this pooling approach, each neighbor's vector is independently fed through a fully-connected neural network; following this transformation, an elementwise max-pooling operation is applied to aggregate information across the neighbor set, which makes it symmetric and trainable.

$$AGGREGATE_k^{pool} = max(\sigma(\mathbf{W}_{pool}\mathbf{h}_{u_i}^k + \mathbf{b}), \forall u_i \in N(V) \quad (15)$$

where max denotes the element-wise max operator and σ is a non-linear activation function.

3) **LSTM Aggregator**: Compared to the mean aggregator, LSTMs have the advantage of larger expressive capability. However, it is important to note that LSTMs are not inherently symmetric (i.e., they are not permutation invariant), since they process their inputs in a sequential manner. We chose to use this aggregator function for our implementation of the GraphSAGE model.

GAT : The GATConv is a foundational pillar of Graph Attention Networks (GAT) [14]. it uses masked self-attention layers to solve the problem of convolution of the current image. The characteristics of neighbor nodes can be aggregated by stacking layers without the need for complex matrix operations nor knowing the entire graph structure upfront.

Computationally, it is highly efficient: the operation of the self-attentional layer can be parallelized across all edges, and the computation of output features can be parallelized across all nodes. No eigendecompositions or similar costly matrix operations are required. The time complexity of a single GAT attention head computing F' features may be expressed as $O(|V|FF' + |E|F')$, where F is the number of input features, and $|V|$ and $|E|$ are the numbers of nodes and edges in the graph, respectively.

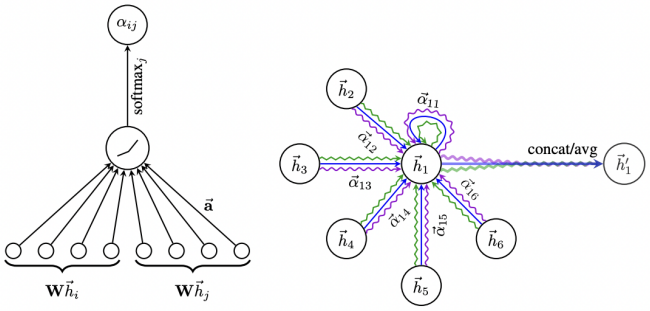

Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}_1'$.

Figure 3: GAT Attention Mechanism [14]

LeGCN : Local Extrema Convolutional layer (LeConv) [9] is an extension of the simple GCN layer that finds the importance of nodes with respect to their neighbors using the difference operator. Internally, a GCN layer computes the importance $\phi = XW$ for each neighbour of a node and subsequently uses a weighted average over all neighbours. If a score of one node is very high, the scores of all its neighbours will be high too, because of the weighted averaging.

On the other hand, LEConv uses the following rule to assign scores to neighbours of a node, which allows the network to select local extremas of neighbour scores and use a better aggregation over neighbourhood [9].

$$\phi_i = \sigma\left(x_i W_1 + \sum_{j \in \mathcal{N}_{(i)}} A_{ij}\left(x_i W_2 - x_j W_3\right)\right) \quad (16)$$

where W_1, W_2, and W_3 are learnable parameters and $\in \mathcal{N}_{(i)}$ is the neighbourhood of node i.

GATv2 : The model is developed by Brody et. al [4], who saw there was room for improvement in GAT model [14]. Since the linear layers (matrix product with the weight matrix and the dot

product with attention coefficients) in the standard GATConv are applied right after each other, they can be collapsed into a single linear operation. As a result, the ranking of attended nodes is unconditioned on the query node. Said otherwise, the ranking (the argsort) of attention coefficients is shared (static) across all nodes in the graph. That's why the expressiveness power of simple GATConv layers is restricted. GATv2Conv fixes this problem by introducing adaptive attention mechanism, that allows the rankings of attention coefficients to differ from one node to another. Due to more flexible attention ratings, the GATv2Conv-based models have a greater expressiveness power and hence perform better.

The exact mechanism of this "fixing" is shown below. In GATConv layers, the dot product of attention coefficients and hidden representations are computed before a non-linearity (usually a Leaky ReLU). In GATv2Conv, the dot product is computed after a non-linearity is applied to a hidden representation.

$$\text{GATConv:} \quad e(\mathbf{h}_i, \mathbf{h}_j) = \sigma\left(\mathbf{a}^T \left[\mathbf{W}\mathbf{h}_i || \mathbf{W}\mathbf{h}_j\right]\right) \tag{17}$$

$$\text{GATv2Conv:} \quad e(\mathbf{h}_i, \mathbf{h}_j) = \mathbf{a}^T \sigma\left(\mathbf{W}\left[\mathbf{h}_i || \mathbf{h}_j\right]\right) \tag{18}$$

Dataset Description

The ogbn-products dataset is an undirected and unweighted graph, representing an Amazon product co-purchasing network. Nodes represent products sold in Amazon, and edges between two products indicate that the products are purchased together.

The data was collected by crawling Amazon website and contains product metadata and review information about 548,552 different products (Books, music CDs, DVDs and VHS video tapes). The dataset was built by Bhatia et al. in 2016 [3].

Each product had a product description text: the text was one-hot encoded, and subsequently Principal Component Analysis (PCA) was applied to reduce the number of features to 100.

The product category is used as the target variable, and the excerpt below shows the some of the categories.

LABEL IDX	PRODUCT CATEGORY
0	Home & Kitchen
1	Health & Personal Care
2	Beauty
3	Sports & Outdoors
4	Books
5	Patio, Lawn & Garden
6	Toys & Games
7	CDs & Vinyl
8	Cell Phones & Accessories
9	Grocery & Gourmet Food

Figure 4: Product

Hyperparameters

Since the dataset was quite large, we used the batch size of 256. We found that a bigger batch size usually resulted in a smoother convergence; however, we were not able to use 512 nodes per batch due to GPU RAM limitations. Also, after 20 epochs, the training set losses and accuracies started to change very little. Therefore, all models were trained for 20 epochs. Moreover, attention-based models (GAT and GATv2)[4][14] had 4 heads, but half as many hidden neurons in each layer as other 3 models. Finally, we used the ELU activation function with the dropout rate of 0.25. (When we used 0.5 as the dropout probability, the training went very slowly because the dropout was introducing too much noise.)

To set other hyperparameters, we ran each model several times to find reasonably good values. The table below presents a summary of finally used hyperparameters.

Hyperparameter Tuning			
Model name	Layers	Hidden Units	Learning Rate
GraphSAGE	3	256	0.002
GCN	3	256	0.003
LeGCN	3	256	0.003
GAT	4	128	0.005
GATv2	4	128	0.004

Mini-Batch Strategy

To construct our mini batches for training, we extracted random subgraphs from our training graph. For a given node, we randomly sampled 20 one-hop neighbours, 15 two-hop neighbours and 10 three-hop neighbours, shuffling the order of nodes in each epoch.

However, when we evaluated accuracy, we used all available neighbours not only in the validation and test sets, but also in the training set. In our opinion, this gives a more accurate reflection of the training set performance compared to model evaluation on the subgraphs with limited number of neighbours.

RESULTS AND DISCUSSION

Exploratory Data Analysis

Overall, the ogbn-products dataset contains 2,449,029 nodes and 123,718,280 edges. We have split them into three categories based on the above-mentioned method to have 196,615 nodes for training, 39,323 nodes for model tuning and validation, and 2,213,091.

Out of the 47 target categories, the most represented classes were Books (668,950 nodes), CDs and Vinyl (172,199 nodes), Toys and Games (158,771 nodes) and Sports and Outdoors (151,061 nodes). The most underrepresented classes were Purchase Circles (29 nodes), Furniture and Decor (9 nodes), and Digital Music (9 nodes).

More than 84% of the nodes had a degree of 1, around 13% had a degree of 2, and only 3% had a degree of 3 or above.

Comparison of Models

For comparing the performance of the models, we used accuracy as a metric. Below, the performances on the training, validation and test sets are given for each model.

Model Accuracies (%)			
Model name	Train	Validation	Test
GraphSAGE	90.09	88.24	78.45
GCN	89.76	88.46	76.64
LeGCN	92.66	90.26	82.59
GAT	91.10	89.42	79.19
GATv2	93.38	91.57	84.27

Our results show that training set and validation set accuracies of different models do not differ significantly. However, their generalization power displays a greater degree of variety. The best performing models were Local Extremum GNN [9] and Dynamic GAT [4]. The former achieved 92.7% accuracy on the training set and 82.6% on the test set. The latter performed marginally better, with 93.4% training accuracy and 84.3% test accuracy. Meanwhile, GraphSAGE [5] achieved a reasonably high accuracy (90.1% on the training set and 78.5% on the test set), despite being a relatively simple model.

If we compare the performances of Local Extremum GNN [9] and Dynamic Graph Attention Networks [4] with their initially proposed versions (Graph Convolutional Networks and static GATs)[7][14], we can see that adaptive aggregation in LeConv layers and adaptive attention mechanism in GATv2 [4] layers do offer a performance advantage: for GCNs, the test set accuracy boost was almost 6%, while it was slightly over 5% for GATs.

We can infer that the reason GAT models [14] perform better than a baseline method like GCN [7] is due to the fact it allows for (implicitly) assigning *different importances* to nodes of a same neighborhood. However, this complexity of GATs is on par with GCNs, since as we said previously, they don't require complex matrix calculations.

Finally, the generalization gap for all models was quite significant. The difference between the training set and test set accuracies was between 9% and 13% for all models. The tightest difference was recorded by the dynamic GAT model [4] (9.1%) and the widest by the GCN model [7] (13.1%). Validation accuracies were usually within 1-2% from the respective training set accuracies. To assess the impact of the chosen split on the models' generalization power, we applied the same split (8% training, 2% validation and 90% test) using stratified sampling. The accuracy was 91.6%, 90.1%, and 88.9% on the training, validation, and test sets, respectively. This training-test accuracy difference of 1.7% implies that the sales-ranking-based split poses a more challenging task than a random stratified split. However, we believe that using this split offers a realistic comparison among models and more closely reflects the real-life generalization power of the above graph neural networks.

CONCLUSION

Over the past few years, graph neural networks have become powerful and practical tools for machine learning tasks in graph domains. This progress owes to advances in expressive power, model flexibility, and training algorithms. In this paper, we replicate several key graph neural networks. For GNN models, we categorize its variants by its propagation modules: 1) Spectral, 2) Spacial, 3) Attention-based Spacial.

Our results show that a model using GATv2Conv layers [4] achieved the highest accuracy, while the one having simple GCN models [7] had the worst performance. Also, attention-based models had a better generalization power, as illustrated by the training accuracy-test accuracy gap, while GCN-layer models had a wider gap between the performances on the two sets.

Also, we discovered that a sales-ranking-based split is a more challenging problem than a random stratified sampling, where models can achieve a very high results (88% test accuracy).

There is still a lot of work to be done in robustness, interpretability, pretraining and complex structure modeling, but we are thankful to have had the opportunity to explore these 5 graph neural network models.

REFERENCES

[1] S. Alessandro and S. Antonina. "Supervised neural networks for the classification of structures." In: *IEEE Transactions on Neural Networks*. (1997).

[2] J. Ankit et al. "Food Discovery with Uber Eats: Using Graph Learning to Power Recommendations." In: *Uber Enginerring* (2019).

[3] K. Bhatia et al. *The extreme classification repository: Multi-label datasets and code*. 2016. URL: http://manikvarma.org/downloads/XC/XMLRepository.html.

[4] Shaked Brody, Uri Alon, and Eran Yahav. "How Attentive are Graph Attention Networks?" In: (2021). URL: https://arxiv.org/abs/2105.14491.

[5] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: (2017). URL: https://arxiv.org/abs/1706.02216.

[6] W. Hu et al. "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: *arXiv preprint arXiv:2005.00687* (2020).

[7] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: (2016). URL: https://arxiv.org/abs/1609.02907.

[8] Stephane Mallet. "A wavelet tour of signal processing". In: (1990). URL: https://www.sciencedirect.com/book/9780123743701/a-wavelet-tour-of-signal-processing.

[9] S. Ranjan E. Sanyal and P. Talukdar. "ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations". In: (2019). URL: https://arxiv.org/abs/1911.07979.

[10] H. Ruining. "PinSage: A new graph convolutional neural network for web-scale recommender systems." In: *Pinterest Engineering* (2018).

[11] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Internal Representations by Error Propagation". In: (1985). URL: https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf.

[12] Franco Scarselli et al. "Supervised neural networks for the classification of structures". In: (2009). URL: https://ieeexplore.ieee.org/document/4700287/.

[13] A Sperduti and A Starita. "Supervised neural networks for the classification of structures". In: (1997). URL: http://refhub.elsevier.com/S2666-6510(21)00001-2/sref198.

[14] Petar Veličković et al. "Graph Attention Networks". In: (2017). URL: https://arxiv.org/abs/1710.10903.