A pipeline for fair comparison of graph neural networks in node classification tasks

Wentao Zhao\textsuperscript{a,b,*}, Dalin Zhou\textsuperscript{c}, Xinguo Qiu\textsuperscript{a}, Wei Jiang\textsuperscript{a}

\textsuperscript{a}College of Mechanical Engineering, Zhejiang University of Technology, Hangzhou, 310023, Zhejiang, China
\textsuperscript{b}School of Intelligent Transportation, Zhejiang Institute of Mechanical & Electrical Engineering, Hangzhou, 310053, Zhejiang, China
\textsuperscript{c}School of Computing, University of Portsmouth, Portsmouth, PO1 3HE, Hampshire, UK

Abstract

Graph neural networks (GNNs) have been investigated for potential applicability in multiple fields that employ graph data. However, there are no standard training settings to ensure fair comparisons among new methods, including different model architectures and data augmentation techniques. We introduce a standard, reproducible benchmark to which the same training settings can be applied for node classification. For this benchmark, we constructed 9 datasets, including both small- and medium-scale datasets from different fields, and 7 different models. We design a k-fold model assessment strategy for small datasets and a standard set of model training procedures for all datasets, enabling a standard experimental pipeline for GNNs to help ensure fair model architecture comparisons. We use node2vec and Laplacian eigenvectors to perform data augmentation to investigate how input features affect the performance of the models. We find topological information is important for node classification tasks. Increasing the number of model layers does not improve the performance except on the PATTERN and CLUSTER datasets, in which the graphs are not connected. Data augmentation is highly useful, especially using node2vec in the baseline, resulting in a substantial baseline performance improvement.

Keywords: graph neural network, node classification, deep learning, topological information

1. Introduction

Many classical models have been applied to perform deep learning studies on data analysis. The convolutional neural network (CNN) and recurrent neural network (RNN) have been used extensively in computer vision (CV) and natural language processing (NLP), respectively. However, when faced with graph-structured data, such as social networking service (SNS) datasets, knowledge graphs, molecular structures, and biological and financial networks, CNNs and RNNs are not easily applied due to the graph structure of the data \cite{1}. To solve problems such as node classification \cite{2} \cite{3}, graph classification \cite{4} \cite{5} \cite{6}, social recommendation \cite{7} \cite{8} and link prediction \cite{9} \cite{10}, graph convolutional network (GCN) models \cite{11}, which exploit message-passing (or equivalently, various neighborhood functions) to precise high levels, have been proposed. More works have been done both optimizes performance of GCNs and solve real life problems. \cite{12} optimizes GCNs from the feature importance perspective via statistical self-attention. \cite{13} put forward a deep learning model, named InfGCN, to identify the most influential nodes in a complex network based on Graph Convolutional Networks. \cite{14} use graph attention temporal convolutional networks to forecast short-term traffic speed. \cite{15} propose a Multimodal Graph Convolutional Networks (MGCN) automatically filter high quality content from a large number of multimedia articles.

Thanks to the establishment of ImageNet \cite{16}, image classification in computer vision has undergone significant development, and many challenging and realistic benchmark datasets have been proposed that facilitate scalable, robust, and reproducible graph machine learning research. The Open Graph Benchmark \cite{17} is one recent benchmark
initiative whose goal is to represent large real-world datasets from various domains; it particularly emphasizes out-of-distribution rationalization performance through valid data splits. A steady trend toward improvement in model accuracy also has been demonstrated on various benchmarks. The accuracy of deep CNNs has developed steadily based on both model architecture improvements and data augmentation [18]. For graph neural networks (GNNs), [19] showed that different data splits lead to different model rankings on node classification tasks, and [20] proposed a fair performance comparison method for GNN architectures. Inspired by the above studies, we constructed a standardized and reproducible experimental environment in this study and used it to test how model architectures and data augmentation techniques impact node classification accuracy. The main contributions of our work are as follows.

- We collect a total of 9 datasets [21] [22] and 7 models that are widely used in various fields in the PyTorch [23] and PyTorch geometric [24] frameworks and use them to compare the expressive power of various model structures on different node classification tasks. For fair comparisons, we use 2 parameter budgets to ensure that the number of parameters and layers is roughly the same. To conduct further research, researchers can easily extend our framework by adding new models with different features and arbitrary datasets from their own experiments.

- We propose a method for fair comparison using the small datasets to evaluate model performance. The existing prior results show that applying small datasets with specific training/evaluation/test splits is unsuitable for making fair model comparisons. Therefore, we specify a 10-fold cross validation split for the small datasets instead of randomly splitting the data several times. We use this approach to build model assessment methods. We perform each experiment several times under the same training conditions to ensure fair comparisons.

- We test some input features to assess the expressive power of embedding by using node2vec and Laplacian vectors as data augmentation techniques to represent the node position and add to the input features. Using this approach, we can evaluate how the various input features affect the results.

The goal of this study is not to find the best-performing GNNs or to obtain the highest classification accuracy (which is computationally expensive) in the specific targets using an extensive hyperparameter grid search. Instead, our goal is to construct a standard benchmark for node classification and a uniform evaluation framework for GNNs. We use numerous GNNs to evaluate the power of the proposed architecture, and future researchers can easily use the framework to compare new proposed models using the existing architectures.

2. Training Procedures

A training process includes mainly datasets, models, and the strategy for updating model parameters. As we can see in the baseline training procedure in Figure 1, Algorithm 1 performs node classification with many graphs, while Algorithm 2 performs node classification for only one graph. For node classification, \( X \) is the embedding of the batch graph whose dimension is \( N \times d \), where \( N \) denotes the number of nodes in the batch graphs, \( d \) denotes the dimension of each node, \( y \) is the node label, and the overall dimension is \( N \times d \), where \( N \) denotes the number of nodes in the batch graphs and \( d \) denotes the dimension of the node label.

The difference between the 2 algorithms is that node classification from only one graph does not require a batch because for some node classification tasks, only one graph is available, and it is difficult to separate the nodes due to their relations. We emphasize that not all node classifications involve only one graph; some have multiple graphs and use minibatches during training. For these tasks, the algorithm process used is similar to Algorithm 1. For datasets that include more nodes and edges, it is easy to exceed the CUDA memory limitations when training models on a GPU; consequently, various methods have been proposed to solve the problems. One is to use a clustering algorithm to separate the graph into clusters and then randomly select some clusters for training, similar to algorithm 1 [25]. The other approach is to use sample subgraphs obtained via a random walk sampling algorithm [26].

2.1. Datasets

The datasets used in our experiments mainly stem from studies [22] [21] [27] that make benchmarks for different tasks. Here, we introduce the 9 datasets summarized in Table 1 including their split schemes, splits, features and...
Algorithm 1: Training a neural network with the Adam optimizer for node classification

**Input**: Node embedding for every graph, initialize(net)

**Output**: Trained model

1. for epoch = 1, 2, ..., K do
2.   for batch = 1, 2, ..., #graphs / b do
3.     graphs ← uniformly random sample graphs;
4.     X, y ← preprocess(graphs);
5.     z ← forward(net, X);
6.     l ← loss(z, y);
7.     grad ← backward(l);
8.     update(net, grad)
9.   end
10. end

Algorithm 2: Training a neural network with the Adam optimizer for node classification

**Input**: Node embedding for every graph, initialize(net)

**Output**: Trained model

1. for epoch = 1, 2, ..., K do
2.   X, y ← preprocess(graphs);
3.   z ← forward(net, X);
4.   l ← loss(z, y);
5.   grad ← backward(l);
6.   update(net, grad)
7. end

Figure 1: A baseline training procedure for node classification

### Table 1: Summary of the datasets used in the experiments. A split strategy, the prediction task and the input features are described for each dataset. Additional details can be found in Section 2.1.

| Name     | #Tasks | Split Scheme | Split       | Node Feat. | Edge Feat. | directed | Metric     |
|----------|--------|--------------|-------------|------------|------------|----------|------------|
| cora     | 1      | 10-fold CV   | 08:01:01    | ✔          | ×          | ×        | Acc        |
| CiteSeer | 1      | 10-fold CV   | 08:01:01    | ✔          | ×          | ×        | Acc        |
| PubMed   | 1      | 10-fold CV   | 08:01:01    | ✔          | ×          | ✔        | Acc        |
| PATTERN  | 1      | random       | 05:01:01    | ✔          | ×          | ×        | Acc        |
| CLUSTER  | 1      | random       | 10:01:01    | ✔          | ✔          | ×        | Acc        |
| Products | 1      | sales rank   | 196615:39323:2213091 | ✔         | ×          | ×        | Acc        |
| arXiv    | 1      | time         | 90941:29799:48603 | ✔         | ×          | ✔        | Acc        |
| MAG      | 1      | time         | 629571:64879:41939 | ✔         | ✔          | ✔        | Acc        |
| Proteins | 112    | species      | 86619:21236:24679 | ✗         | ✔          | ✗        | ROC-AUC    |

metrics. More details (such as the classes, domains, numbers of nodes and edges, and dimensions of the input features) are shown in Table A.6 (Appendix A).

**Cora, CiteSeer** and **PubMed**: For the node classification tasks, we first executed our experiment on three widely used benchmark datasets: Cora, CiteSeer and PubMed [27] [28], which are all citation networks.

**Prediction task**: For the Cora dataset, the task is to predict the subject of the paper (node) based on the surrounding node data and the graph structure. The different subjects are ‘Case Based’, ‘Genetic Algorithms’, ‘Neural Networks’, ‘Probabilistic Methods’, ‘Reinforcement Learning’, ‘Rule Learning’, and ‘Theory’. For the CiteSeer dataset, the task
is to predict which domain the paper (node) belongs to. All the papers are grouped into 6 classes: ‘Agents’, ‘AI’, ‘DB’, ‘IR’, ‘ML’ and ‘HCI’. The publications from the PubMed dataset pertaining to diabetes are classified into one of three classes: ‘Diabetes Mellitus, Experimental’, ‘Diabetes Mellitus Type 1’, or ‘Diabetes Mellitus Type 2’.

Dataset splitting: An earlier work [19] compared the performance of various GNNs on node classification tasks and found that choosing different training/validation/testing splits leads to different performance rankings. This result occurs primarily because the datasets used were too small to fully leverage the power of data-hungry GNNs. For fair comparisons, a 10-fold cross-validation split is used to train the models, similar to reference [20]. Ten sets of training, validation and testing data indices at a ratio of 8:1:1, respectively, are used to perform node classifications and calculate the test accuracy. Figure 2 reports the pseudocode of the entire process used to obtain the results of the classification accuracy, and the overall procedure is visually summarized in Figure A.4. Note that the model training processes are the same for Cora, CiteSeer and PubMed and that the datasets are represented later. We introduce the experimental setting in detail below.

**PATTERN** and **CLUSTER** are both artificially generated datasets generated with stochastic block models [29]. These datasets are widely used to model communities by modulating intra and extra community connections to control the difficulty of tasks in social networks, and they can be used for node classification tasks. **PATTERN** and **CLUSTER** contain a total of 14 K and 12 K graphs, respectively. The distribution of the number of nodes in these graphs is depicted in Figure A.2.

Prediction task: **PATTERN** [30] is used to recognize specific predetermined subgraphs. For all the data, a graph \( G \) is generated with 5 communities whose sizes are randomly selected between [5,35]. We then randomly generate 100 patterns \( P \) composed of 20 nodes. The node features for \( P \) and \( G \) are generated as a random signal with values of 0,1,2. The graph sizes are 44–195. The output node labels have a value of 1 if the node belongs to \( P \) and 0 if it belongs to \( G \). **CLUSTER** is targeted at identifying community clusters. We generate 6 SBM clusters with randomly selected sizes in the range [5,35]. The graphs contain 40–190 nodes. Each node is embedded with a value from 0,1,2,...,6. If the value of the node is \( a(a > 0) \), the node belongs to class \( a−1 \), and if the value is 0, the node class is unknown. Only one labeled node is randomly assigned to each community, and most of the features are set to 0. When calculating accuracy, we use weighted accuracy w.r.t. the class sizes.

Dataset splitting: The **PATTERN** and **CLUSTER** datasets are both split randomly. The **PATTERN** dataset split includes 10,000 training, 2,000 validation, and 2,000 testing graphs, while the **CLUSTER** dataset is split into 10,000 training, 1,000 validation, and 1,000 testing graphs. We save the splits and subsequently use them to ensure fair comparisons. In addition to the artificially generated datasets, we import some datasets from the open graph benchmark dataset (OGB) [17]. The graphs all stem from real-world tasks. Unlike the datasets discussed above, these datasets include only one graph.

**Products**: The Products dataset is an undirected and unweighted graph. It is an Amazon product co-purchasing network [31]. The node features and the edges between 2 nodes are represented by a dimensionality-reduced bag-of-words of the product descriptions and the products that are purchased together, respectively, following [25].

Prediction task: The task is to predict which category the product belongs to. Forty-seven top-level categories are used as the target labels.

Dataset splitting: The previously used Cora, CiteSeer, and PubMed citation networks are typically split randomly; however, instead of performing a random split, [17] split the dataset by sales ranking. The top 8%, the next top 2% and the remainder of the records are used for training, validation and testing, respectively. Thus, as in real-world scenarios, we label the most popular products and leave the less popular products for validation and testing. This is a more realistic segmentation approach.

**arXiv**: arXiv is a directed graph that represents the citation network between all computer science (CS) arXiv papers [32]. The nodes represent an arXiv paper embedded with a 128-dimensional feature vector obtained by averaging the embeddings of the words in its title and abstract.

Prediction task: The task is to predict the subject areas to which an arXiv CS paper belongs (e.g., cs.AI, cs.LG, and cs.OS). The dataset includes forty labels that were manually determined by the paper’s authors and arXiv moderators.

Dataset splitting: We use a realistic data split based on the publication date of the papers to split this dataset; thus, papers published through 2017 are used for training, and papers published in 2018 and 2019 are used as validation and testing datasets, respectively.

**MAG**: MAG is a heterogeneous network composed of a subset of the Microsoft Academic Graph (MAG) [32]. It consists of 4 entity types: papers (736,839 nodes), authors (1,134,649 nodes), fields of study (59,965 nodes) and
Algorithm 3: Model Assessment (k-Fold CV)

Input: Node embedding for graph D, initialize(net)
Output: Test Acc.

1. Split D into k folds F₁,...,F_k;
2. for i = 1,2,...,k do
   3. trainᵢ, validᵢ, testᵢ ← (∪ⱼ∈Fⱼ), Fᵢ;
   4. for z = 1,2,...,R do
      5. modelₖ ← Train(trainᵢ, validᵢ);
      6. pᵢ ← Eval(modelₖ, testᵢ);
   7. end
   8. perfᵢ ← ∑ᵢ=1^R pᵢ/R;
end
10. return ∑ᵢ=1^k perfᵢ/k

Algorithm 4: Model Training

Input: trainᵢ, validᵢ, initialize(net)
Output: Trained model

1. for epoch = 1,2,...,N do
   2. model ← Train(trainᵢ);
   3. lossvalᵢ ← Eval(model, validᵢ);
   4. while lossvalᵢ > lossvalᵢ₋₁ do
      5. patience +=
      6. if patience >= lr_schedule_patience then
         7. lr = lr × 0.5;
         8. patience = 0;
      end
   7. end
   8. if lr <= minlr or time > maxtime then
      9. break
   10. end
14. end
15. return model

Figure 2: Pseudocode for model assessment (top) and model training (bottom). In Algorithm 3, ‘Train’ refers to Algorithm 4, whereas a strategy is introduced to train the models according to the results of the inference phases. The models are then used to test the classification accuracy in the test_k part and obtain the pᵢ. We train the models and test them R times and obtain the average perfᵢ. Finally, the experiments are used in the k-fold CV, and the average results are obtained.

Institutions (8,740 nodes). In addition, 4 types of directed relations connect 2 of the entity types. Each author is affiliated with an institution, an author writes a paper, the paper cites other papers, and each paper has a primary topic field of study. A 128-dimensional word2vec feature vector is used as the paper node embeddings. The other node types are not associated with the input node features.

Prediction task: The task is to predict the venue (conference or journal) at which each paper was presented. Based on the data, the papers could theoretically have been presented at any of 349 different venues, making the task a 349-class classification problem. To simplify the task, we consider only the paper nodes and the types of directed relations connecting 2 entity types, namely, a paper citing another paper, to transfer the task to be dealt with by an isomorphic graph.

Dataset splitting: Similar to arXiv data, we consider a realistic data split based on the paper publication dates. In practice, we always train on older existing papers and test on newer published papers. Finally, we used the papers
published through 2017 for training. The papers published in 2018 and 2019 are used as the validation and testing datasets, respectively.

**Proteins:** Proteins consist of undirected, weighted, and typed graphs where the nodes represent proteins, and the edges indicate different types of biologically meaningful associations between proteins \[33\] \[34\]. All the edges are embedded with 8-dimensional features. Each dimension represents the strength of a single association type and takes a value between 0 and 1. The dataset includes a total of 8 association types, constituting 8-dimensional features. In contrast to the other datasets, the proteins do not have node features; they possess only edge features—but there are more than 30 million edges. To better train the model using the input features, we use the averaged edge features of incoming edges as node features.

Prediction task: The task is to predict whether various protein functions are present. In total, there are 112 possible labels to predict. The result is measured by the average of the ROC-AUC scores across the 112 tasks.

Dataset splitting: We split the protein nodes according to the species from which the proteins come. This makes it possible to evaluate model generalizability across different species.

2.2. Embedded Information Regarding the Datasets

Using Laplacian eigenvectors as positional embeddings: For a graph, the positional features that represent each node are important, especially when working with graphs that exhibit symmetry in their structures, such as node or edge isomorphism \[35\] \[36\]. Laplacian eigenvectors \[37\] are a technique that embeds graphs into Euclidean space. The eigenvectors are used to build a meaningful local coordinate system that preserves the global graph structure. The eigenvectors are defined via factorization of the graph’s Laplacian matrix.

\[
\Delta = I - D^{-1/2}AD^{-1/2} = U^T \Lambda U \tag{1}
\]

The Laplacian eigenvectors of a graph can be calculated by Laplace decomposition, where \(A\) is the \(n\times n\) adjacency matrix, \(D\) is the degree matrix, \(\Lambda\) is the eigenvalue and \(U\) is the eigenvector. \(U\) is a type of positional encoding; the \(k\)-smallest nontrivial encoding is used as a supplement to the node embeddings to enrich the input node information. Laplacian eigenvectors are used to provide smooth encoding coordinates for neighboring nodes.

Node2vec \[38\]: In NLP tasks, word2vec \[39\] is a commonly used word embedding method that describes the cooccurrence relationships between words in sentence sequences in the corpus and then learns the vector representation of words. The idea behind node2vec is similar to word2vec; it uses the node co-occurrences in the graph to learn the vector representations of nodes. In contrast to deepwalk \[8\] or randomwalk to obtain the nearest neighbor sequence of vertices, node2vec uses a biased random walk. It is an unsupervised node embedding method.

2.3. Model Selection

To ensure fair model comparisons, a parameter budget for the models is controlled in our experiments. The pipeline of experiments is illustrated in Figure 3 for all GNNs. First, an embedding layer is used in the input layer to satisfy the input requirements of the GNN layers. GNN layers are then used in the \(L \times \text{GNN}\) layers as well as the batch norm and activation function. Finally, MLP is used to embed the graph.

Most GNN implementations involve a GCN-Graph Convolutional Network \[11\], GAT-Graph Attention Network \[40\], GraphSAGE \[41\], GIN–Graph Isomorphism Network \[42\], MoNet–Gaussian Mixture Model Network \[43\], Residual Gated Graph Convolutional Network \[44\], and Gated Graph Convolutional Network \[45\]. All these model architectures are explained in Appendix B.

3. Experiments

3.1. Data Splitting and Model Assessment

On small datasets such as Cora, CiteSeer and PubMed, we use the \(k\)-fold cross-validation split for model assessment. As shown in Algorithm 3 (Figure 2), we first split the datasets into \(k\) folds \((k = 10)\). Note that the class proportions are preserved in all the data splits (training, validation, and testing splits). All the data partitions are preprocessed; thus, the models are trained and evaluated on the same data splits. We train the models \(R\) times \((R = 3)\) on each training and validation fold and obtain an average accuracy \(\text{perf}_k\) for each fold. The average of the \(k\) folds is used
as the final accuracy result of the classification. Finally, we execute each model 4 times with different seeds and use the mean value as the final classification result. On the other datasets, the respective strategies for splitting the datasets are introduced above; each model is executed 4 times, and the mean accuracy is taken as the final classification result.

3.2. Training

The goal of this study is not to find an optimal set of hyperparameters to obtain the best model performance but rather to compare the power of models within a parameter budget. For the same datasets, we use the same hyperparameters to train the models for node classification. A budget of parameters—including different numbers of layers—is used to help ensure a fair model comparison; the details can be seen in the results. The model training process strategy is the same for all the datasets, and the value of $N$ is set to 1,000 (Algorithm 4 in Figure 2). If the inference results lossval$_n$ are not decreased lr_schedule_patience times, the learning rate will decrease by half. We implement early stopping when $lr = min_lr$ or $time > max_time$. For the loss function, all use the CrossEntropyLoss except the proteins, which use the BCEWithLogitsLoss because of the special classified content. The Adam optimizer [46] for all models is used in the experiments, and the hyperparameters, e.g., initial learning rate, patience times, min$lr$, and max$lr$, are shown in the code.

4. Results and Discussion

We perform the experiments on 3 planetoid datasets, Cora, CiteSeer and PubMed. The model training and model assessment methods are shown in Section 3 (Experiments), and each dataset has 2 parameter budgets. Due to the different dimensions of the input features, different numbers of parameter budgets are used for each dataset. For Cora, CiteSeer and PubMed, 260k/460k, 550k/750k and 150k/350k parameters are used, respectively. The small parameter budget has 4 layers, while the larger budget has 16 layers. The experimental pipeline for GNNs is shown in Section 2.3 (Model selection). Our goal is to conduct a fair comparison of the different model architectures on small datasets using the same settings.

Table 2 shows the results of our experiments. We find that for deeper layers, the results sometimes do not improve. As the model depth increases, GNNs tend to suffer performance degeneration. The main reasons for this phenomenon are overfitting, oversmoothing [47] and vanishing gradients. Importantly, for the Cora and CiteSeer datasets, all the GNNs clearly outperform the MLP. The results suggest that the GNNs can actually exploit the topological information of the graphs in the dataset. However, for PubMed, the power of the topological information of the graphs is inconspicuous. Due to the small datasets, the results are easily overfitted. Most GNNs reach the accuracy of the training dataset near 100%. For all the models, GIN with 16 layers performs best in 2 small datasets and performs well in the other small dataset. For the baseline, MLP, an overly parameterized baseline, is not able to overfit the training data completely, even worsening. The baseline results have a higher standard deviation than the GNNs. This shows the training instability of the MLP.

Table 3 shows the results on the PATTERN and CLUSTERZ datasets, which are widely used in social networks. Note that they are weighted accuracies w.r.t. the class sizes, and the same training hyperparameters are used during training for a fair comparison. For PATTERN and CLUSTER, 2 classes and 6 classes exist, respectively (see Table A.6).
Table 2: Performance on the Planetoid datasets with 10-fold cross-validation. All the experiments were executed 4 times using the same hyperparameters with different seeds, and the mean ± standard deviation of the results was calculated. The highest value and good values are indicated in red and blue respectively.

| dataset     | Model          | L. params | Train Acc. | Val Acc. | Test Acc. | Epoch |
|-------------|----------------|-----------|------------|----------|-----------|-------|
| Cora        | MLP            | 4          | 260167     | 90.877±5.5075 | 54.480±2.4206 | 53.627±4.8318 | 217.9±3.2939 |
|             | GCN            | 4          | 261191     | 99.996±0.0008 | 85.467±0.1454 | 84.989±0.119 | 225.658±0.9539 |
|             | GraphSage      | 4          | 462047     | 99.960±0.0092 | 85.516±0.2035 | 84.653±0.191 | 226.551±0.9809 |
|             | ResGatedGCN    | 4          | 263673     | 100.0±0.0   | 85.947±2.2164 | 85.510±0.2481 | 216.233±0.9416 |
|             | GatedGCN       | 4          | 466625     | 100.0±0.0   | 86.364±0.4295 | 85.696±0.3015 | 211.233±0.2108 |
|             | GAT            | 4          | 264387     | 99.996±0.0008 | 85.922±0.4375 | 85.389±0.6281 | 222.583±0.9492 |
|             | GIN            | 4          | 467887     | 99.973±0.0054 | 86.156±0.2771 | 85.752±0.3085 | 216.908±1.113  |
|             | Monet          | 4          | 268739     | 99.962±0.017 | 86.337±0.5164 | 86.204±0.341 | 237.75±3.014 |
|             | MNIST          | 4          | 465219     | 98.128±0.1248 | 85.885±0.2054 | 84.838±0.2387 | 382.783±2.8856 |
|             | GIN            | 4          | 262215     | 99.990±0.0045 | 85.568±0.0714 | 85.195±0.1021 | 215.825±0.5494 |
|             | GIN            | 4          | 466503     | 99.959±0.0078 | 85.981±0.1752 | 85.149±0.2009 | 213.516±0.2043 |
|             | GIN            | 4          | 261069     | 100.0±0.0   | 87.038±0.1481 | 86.739±0.0972 | 214.041±0.1642 |
|             | GIN            | 4          | 461176     | 99.991±0.0051 | 87.054±0.2471 | 87.032±0.1584 | 225.133±0.7488 |
|             | GIN            | 4          | 265808     | 100.0±0.0   | 85.679±0.3681 | 85.413±0.3714 | 233.134±2.6961 |
|             | Monet          | 4          | 464012     | 100.0±0.0   | 85.624±0.491 | 85.287±0.5702 | 229.367±3.1476 |
| PubMed      | MLP            | 4          | 550694     | 98.185±1.119 | 56.198±2.3063 | 55.359±2.2809 | 212.0±0.9518 |
|             | GCN            | 4          | 748838     | 51.401±1.2994 | 35.454±0.6745 | 35.347±0.8122 | 222.675±1.5597 |
|             | GraphSage      | 4          | 551178     | 99.947±0.0041 | 70.367±0.045 | 70.311±0.0943 | 214.325±0.7416 |
|             | GraphSage      | 4          | 752934     | 99.712±0.0565 | 70.500±0.2185 | 70.790±0.196 | 207.196±6.2345 |
|             | ResGatedGCN    | 4          | 547381     | 99.977±0.0   | 72.832±0.0484 | 72.689±0.1979 | 208.603±0.25 |
|             | GatedGCN       | 4          | 751812     | 99.977±0.0   | 72.022±0.3888 | 71.995±0.3202 | 201.766±0.9031 |
|             | GatedGCN       | 4          | 549534     | 99.977±0.0   | 72.783±0.7475 | 72.662±0.7417 | 211.566±0.6492 |
|             | GatedGCN       | 4          | 758467     | 99.977±0.0   | 72.571±0.462 | 72.516±0.3692 | 205.741±3.304 |
|             | GatedGCN       | 4          | 549282     | 99.975±0.0051 | 73.131±0.1474 | 72.386±0.0525 | 221.508±0.747 |
|             | GAT            | 4          | 753285     | 99.584±0.0472 | 72.44±0.415 | 71.468±0.2711 | 207.825±0.396 |
|             | GAT            | 4          | 552742     | 99.923±0.0076 | 70.490±0.1094 | 70.920±0.1353 | 206.358±0.5209 |
|             | GIN            | 4          | 757030     | 99.789±0.0195 | 70.758±0.2116 | 71.091±0.1623 | 199.875±0.9472 |
|             | GIN            | 4          | 549232     | 99.974±0.0016 | 73.308±0.1784 | 73.966±0.0609 | 206.066±0.4422 |
|             | Monet          | 4          | 752038     | 99.964±0.0048 | 74.024±0.0876 | 74.522±0.0924 | 210.4±0.2341 |
|             | Monet          | 4          | 552277     | 99.977±0.0   | 70.790±0.5502 | 70.628±0.4377 | 218.116±0.5815 |
|             | Monet          | 4          | 757010     | 99.977±0.0   | 70.387±0.8566 | 70.016±0.5957 | 211.6±3.5057 |

Appendix A). From the baseline, MLP, in which the topological information of the graphs is not used in the models, there are only near 50% and 21% in the classification, just near-random guesses. Even when the layers become deeper, the results do not improve. All the GNNs except GatedGCN obtain better performance using the graphic entity. There is no overfitting in the models, and as the GNNs become deeper, the results have a large boost (except for GatedGCN, the results have a large variance when training 4 times, and when training in CLUSTER in 32 layers, the results obtain only 20.9217%, just near the baseline). This is especially true in CLUSTER.
the input features a classification accuracy. Here, we adopt the Node2vec embeddings and Laplacian eigenvectors to investigate how does not achieve better performance. The layers become deeper, the MLP degrades on all 4 datasets. It seems that for the baseline, a large parameter budget each layer of the network, performs the worst—even worse than the baseline in Products when using 16 layers. As because MAG includes more classes than arXiv, performing classification in MAG is more difficult. Among these 4 datasets, the models GAT and GCN perform well compared to the others, while GIN, which uses feature outputs from each layer of the network, performs the worst—even worse than the baseline in Products when using 16 layers. As the layers become deeper, the MLP degrades on all 4 datasets. It seems that for the baseline, a large parameter budget does not achieve better performance.

Table 3: Performance on the SBM datasets for node classification. All the experiments were executed 4 times using the same hyperparameters with different seeds, and the mean ± standard deviation of the results was calculated. Red: the best model, Blue: good models.

| dataset | Model       | L | params     | Train Acc.       | Val Acc.         | Test Acc.        | Epoch |
|---------|-------------|---|------------|------------------|------------------|------------------|-------|
| PATTERN | MLP         | 4 | 105263     | 50.2716±0.0132  | 50.501±0.0      | 50.505±0.0       | 42.0±0.0 |
|         | GAT         | 16| 100923     | 85.5757±0.038   | 85.479±0.0136   | 85.606±0.037     | 81.0±6.8313 |
|         | GraphSage   | 16| 500823     | 86.1307±0.0542  | 85.511±0.0607   | 85.6337±0.0443   | 67.25±3.4034 |
|         | ResGatedGCN | 4 | 106123     | 61.5527±0.0265  | 60.7374±0.0715  | 60.8821±0.0196   | 89.5±8.8129  |
|         | GatedGCN    | 16| 499887     | 63.849±0.0858   | 62.8452±0.1194  | 62.7989±0.0528   | 92.75±11.7011 |
|         | GIN         | 16| 104003     | 84.7557±0.2293  | 84.7014±0.217    | 84.8624±0.2249   | 78.5±10.3441 |
|         | MoNet       | 16| 502223     | 86.3522±0.0901  | 85.4158±0.0652  | 85.6615±0.0625   | 63.0±10.1653 |
|         | 8           | 106644     | 58.8313±10.3866 | 58.8632±10.265  | 58.8123±10.1709  | 62.25±18.3371 |
|         | MLP         | 4           | 106015     | 20.9432±0.0016  | 20.9954±0.0045  | 20.937±0.0087    | 43.0±2.0   |
|         | GAT         | 16          | 507961     | 20.944±0.0048   | 20.9986±0.0018  | 20.9282±0.0038   | 68.75±11.1467 |
|         | GraphSage   | 16          | 101655     | 49.743±3.5857   | 49.365±3.2667   | 48.8465±3.5681   | 63±4.5461  |
|         | ResGatedGCN | 4           | 106187     | 72.5567±3.7478  | 76.4692±2.0717  | 76.6036±2.3658   | 76.0±11.7473 |
|         | GatedGCN    | 16          | 500503     | 63.0462±0.0499  | 62.2744±0.1038  | 62.4846±0.0369   | 80.23±11.6154 |
|         | GIN         | 16          | 104355     | 62.3735±0.882   | 61.5646±0.7769  | 61.3884±0.7836   | 106.0±21.8021 |
|         | MoNet       | 16          | 502615     | 87.4099±0.9555  | 73.4934±1.1984  | 73.56±0.292      | 60.25±3.7749 |
|         | 8           | 107072     | 36.3052±8.6864 | 36.4263±8.6648  | 36.1752±8.6751   | 94.25±12.5797 |
|         | GatedGCN    | 32          | 503911     | 20.9779±0.0587  | 20.9943±0.1439  | 20.9217±0.2132   | 45.25±2.2174 |
|         | GAT         | 4           | 100122     | 58.8198±0.2584  | 58.0899±0.1739  | 57.9182±0.2563   | 69.9±4.0825  |
|         | GIN         | 16          | 530690     | 79.196±0.4774   | 70.999±0.2952   | 71.061±0.3584    | 70.5±9.3274  |
|         | MoNet       | 16          | 517570     | 66.4919±1.2986  | 64.7361±1.1545  | 64.612±1.184     | 80.25±3.8622 |
|         | 4           | 108178     | 59.4267±0.5389 | 58.8536±0.443   | 58.6417±0.5125   | 73.25±7.719    |
|         | MoNet       | 16          | 528883     | 75.792±2.1493   | 72.7219±1.3878   | 72.8281±1.3572   | 68.5±3.5229  |

Unlike the datasets for node classification, which have only 1 graph, PATTERN and CLUSTER have 14k and 12k graphs, respectively. The task is to classify the nodes in each graph. It seems that for these tasks, oversmoothing is not a serious problem because not all the nodes are connected together. It can also be found in the results that the accuracy of the classification improves when the layers become deeper. MoNet with 16 layers achieves the best performance.

The OGBN dataset in Table 4 was collected entirely from the Open Graph Benchmark [17]. The recent minibatch-based GNNs are always used in OGBN because there is only 1 graph, and these graphs can easily exceed the available CUDA memory during training. Before training using the different model architectures, we first randomly sample the nodes within a graph and return their induced subgraph to separate the dataset for all the datasets except arXiv (for arXiv, when choosing the appropriate parameters, the CUDA memory is sufficient to address the full-batch version of GNNs).

We can find that both GNNs perform well on arXiv and MAG, which both come from academic graphs; however, because MAG includes more classes than arXiv, performing classification in MAG is more difficult. Among these 4 datasets, the models GAT and GCN perform well compared to the others, while GIN, which uses feature outputs from each layer of the network, performs the worst—even worse than the baseline in Products when using 16 layers. As the layers become deeper, the MLP degrades on all 4 datasets. It seems that for the baseline, a large parameter budget does not achieve better performance.

Numerous methods for data augmentation have been proposed in the image classification field that can increase the classification accuracy. Here, we adopt the Node2vec embeddings and Laplacian eigenvectors to investigate how the input features affect the accuracy (see Table 5). The object datasets, including arXiv and Products, are used to test how the input features affect accuracy with the MLP, GCN and GAT models. We use MLP as the baseline and include
| dataset | Model | L | params | Train Acc. | Val Acc. | Test Acc. | Epoch |
|---------|-------|---|--------|-----------|----------|-----------|-------|
| Products | MLP   | 4 | 89887 | 80.41±0.3601 | 75.87±0.2874 | 70.51±0.2042 | 93.25±12.566 |
|         |       |   | 301357 | 75.225±0.3854 | 71.47±0.2145 | 68.96±0.324 | 89.5±9.574 |
|         | GCN   | 4 | 90967 | 85.988±0.3475 | 82.489±0.3126 | 77.989±0.1991 | 82.45±19.261 |
|         |       |   | 301204 | 86.175±0.4111 | 83.016±0.2975 | 76.4±0.7194 | 71.25±12.842 |
|         | GAT   | 4 | 89978 | 76.827±0.0457 | 69.52±0.1848 | 68.71±0.064 | 52.3±20.14 |
|         |       |   | 304967 | 77.37±0.09 | 74.75±0.1367 | 68.95±0.1465 | 57.0±3.9158 |
|         | ResGCN | 4 | 88909 | 84.87±0.5857 | 81.427±0.6523 | 78.1±0.856 | 55.2±7.932 |
|         |       |   | 304132 | 87.63±0.4248 | 83.76±0.1287 | 79.71±0.2037 | 229.75±17.131 |
|         | GCN   | 4 | 90803 | 72.02±0.3.3052 | 65.697±0.7595 | 62.4±0.8142 | 45.7±1.7321 |
|         |       |   | 305897 | 58.11±0.2294 | 53.427±0.6866 | 53.79±1.1951 | 54.5±3.166 |
|         | GAT   | 4 | 93272 | 83.492±0.6073 | 79.86±0.5536 | 75.10±0.388 | 46.3±2.0156 |
|         |       |   | 323384 | 85.467±0.4178 | 84.52±0.4513 | 77.0±0.622 | 45.7±2.3805 |
|         | ResGCN | 4 | 90039 | 47.19±0.5131 | 45.18±0.0908 | 47.2±0.1383 | 44.4±2.1983 |
|         |       |   | 299376 | 49.304±0.5463 | 49.12±0.7496 | 46.5±1.7343 | 44.7±2.0156 |
|         | GCN   | 4 | 92704 | 82.18±0.2751 | 79.77±0.1061 | 74.80±0.1685 | 45.7±2.3805 |
|         |       |   | 299019 | 83.342±0.303 | 79.58±0.3526 | 75.06±0.8849 | 49.25±5.9121 |

GCN and GAT because all these models perform well on these 2 tasks. MLP in particular achieves a substantial im-
Table 5: Performance on the OGBN datasets including arXiv and Products. All the experiments were executed 4 times using the same hyper-parameters with different seeds, and the mean ± standard deviation of the results was calculated. The indicators that improved significantly are highlighted in red, while the merely improved indicators are highlighted in blue. Models with the suffix -pe use Laplacian eigenvectors as node positional encodings with a dimension of 64. Models with the suffix -ne use node2vec embeddings in the input features.

| Dataset | Model   | L | params | Train Acc. | Val Acc. | Test Acc. | Epoch |
|---------|---------|---|--------|------------|----------|-----------|-------|
| arXiv   | MLP     | 4 | 98807  | 92.88 ±0.6601 | 73.88 ±0.2464 | 73.72 ±0.0378 | 77.25 ±0.909 |
|         | MLP-pe  | 16| 300479 | 83.3975 ±0.3577 | 70.05 ±0.1252 | 56.08 ±0.1723 | 78.75 ±6.8496 |
|         | MLP-ne  | 16| 903095 | 83.1328 ±0.3495 | 69.9654 ±0.5337 | 55.8319 ±0.4602 | 78.5 ±1.962 |
|         | GCN     | 16| 106707 | 93.7025 ±0.0969 | 89.2162 ±0.086 | 70.9476 ±0.3354 | 68.5 ±2.3805 |
|         | GAT     | 16| 317375 | 92.9284 ±0.1573 | 87.5626 ±0.2582 | 66.9452 ±0.446 | 68.0 ±5.7155 |
|         | GCN-ne  | 16| 40863 | 92.4537 ±0.0749 | 90.9016 ±0.1326 | 75.2748 ±0.1767 | 118 ±13.3292 |
|         | GAT-pe  | 16| 302999 | 92.6767 ±0.0947 | 91.6687 ±0.075 | 76.018 ±0.1216 | 108.75 ±2.5 |
|         | GAT-ne  | 16| 308184 | 92.7685 ±0.1377 | 91.2646 ±0.1161 | 76.1586 ±0.1262 | 121.75 ±9.7767 |
|         | GCN     | 16| 107759 | 93.5741 ±0.1473 | 91.7135 ±0.0884 | 75.000 ±0.3371 | 85.0 ±10.4243 |
|         | GCN-ne  | 16| 317067 | 93.6397 ±0.1337 | 91.772 ±0.0534 | 74.6966 ±0.2146 | 94.75 ±9.4296 |
|         | GAT     | 16| 96879 | 91.595 ±0.0995 | 90.21 ±0.0408 | 76.43 ±0.1485 | 148.75 ±12.816 |
| Products| MLP     | 4 | 94443  | 92.4556 ±0.1256 | 90.8343 ±0.1218 | 75.3129 ±0.1286 | 127.5 ±13.0767 |
|         | MLP-pe  | 16| 207519 | 91.5923 ±0.0928 | 90.3702 ±0.0941 | 76.4812 ±0.1802 | 156.75 ±16.5202 |
|         | MLP-ne  | 16| 299695 | 92.3608 ±0.0987 | 90.8387 ±0.1179 | 77.3693 ±0.3204 | 144 ±0.5461 |
|         | GAT     | 16| 114287 | 93.2509 ±0.1768 | 91.4738 ±0.0939 | 76.2792 ±0.2283 | 104.25 ±11.8708 |
|         | GAT-ne  | 16| 307759 | 93.4481 ±0.2017 | 91.6169 ±0.1513 | 76.259 ±0.3628 | 116 ±10.2307 |

Improvement on 2 of the datasets with the node embedding features. It appears that the node embedding well represents the topological information in the input features, causing a large effect. When using the node embedding on arXiv, the GNNs still achieve improvements; it seems that input features containing the topological information still have effects on arXiv. However, when the GNNs are applied to the Products dataset, no improvements are found. All the models with Laplacian eigenvectors (except MLP with 16 layers) achieve improvements on Products, but only GCN and GAT with 4 layers improve on arXiv. This phenomenon shows that the impact of the location information described by the Laplacian eigenvectors plays a larger role in the Products dataset than in the arXiv dataset.

5. Conclusions

This study created a benchmark framework for node classification with GNNs, allowing researchers to test how the architectures and input features of various models affect the results. A k-fold model assessment was defined and applied to the small datasets. We also defined a set of model training procedures, including how to decrease the learning rate and when to terminate training. The result is a standard experimental pipeline for GNNs that helps ensure fair model comparisons. For all the models, 2 parameter budgets with different numbers of layers are used to
test whether the number of parameters affects the results. Overall, our goal in this study was to construct a benchmark to assist in testing model performance on different datasets instead of obtaining the best classification accuracy.

- We found that for graphs containing topological information, all the GNNs performed well compared with the baseline on the small datasets except for PubMed. It appears that topological information does not play an important role in improving GNN classification accuracy on PubMed.

- For the artificially generated SBMs used in social networks, we found that increasing the number of layers elevates the results, especially on CLUSTER for most models. It appears that oversmoothing is not a serious problem in node classification tasks on datasets that contain many unconnected graphs.

- To accommodate CUDA memory limitations, we first randomly sample nodes within a graph and return their induced subgraph to separate the dataset into OGBN datasets (e.g., MAG, Products and Proteins). The results show that GCN and GAT perform best on the OGBN datasets and indicate that the traditional classical models are still highly useful.

- We applied some data augmentation methods in our experiments, including node2vec embeddings and Laplacian eigenvectors. We found that the use of node2vec embeddings in MLP provides a substantial improvement on the arXiv and Products datasets and improved the GNN performance on arXiv but did not work with GNNs on the Products dataset. In contrast, the Laplacian eigenvectors did help on Products, but the improvement was limited.

### 5.1. Future Work

Methods such as dropedge [48], pairnorm [49] and nodenorm [50] have been proposed to solve the problems of oversmoothing and overfitting. However, these methods were tested only on small datasets in the original studies. Thus, their applicability to large datasets has not been fully verified. The experimental conditions with respect to the split among training/validation/testing and model assessment do not seem to ensure a fair comparison. Therefore, in future work, we will try to use our benchmark to test the role of the above methods in reducing the convergence speed, oversmoothing, and overfitting. When working with directed graphs, we first add edges to make them undirected, and it will be fruitful to explore how edge direction information can be considered to improve prediction performance. In addition, some datasets include some node temporal information (e.g., the year in which papers are published in arXiv and MAG); this temporal information could be included in the input features to improve the classification accuracy. To accommodate CUDA memory limitations, for some datasets, we only randomly sampled the graph nodes and returned an induced subgraph to partition the dataset. Some minibatch GNNs, especially NeighborSampling [41], GraphSAINT [26] and ClusterGCN [25], are used to perform node classification. Sometimes, these even slightly outperform the full-batch version of GNNs, which does not fit into ordinary GPU memory. Our benchmark framework is also suitable for testing these models.

### Appendix A.

Table A.6: Summary of the datasets used in experiments. A split strategy, prediction task and input features are described for each dataset; more details can be found in Section 2.1.

| Domain                               | Name   | #graphs | #nodes*dim | #edges*dim | Classes |
|--------------------------------------|--------|---------|------------|------------|---------|
| citation network datasets            | Cora   | 1       | 2708*1433  | 5278       | 7       |
|                                      | CiteSeer | 1       | 3327*3703  | 4552       | 6       |
|                                      | PubMed | 1       | 19,717*500 | 44,324     | 3       |
| Mathematical Modelling in social networks | PATTERN | 14k     | 44-195*1   | 752-15900  | 2       |
|                                      | CLUSTER | 12k     | 43-190*1   | 524-10752  | 6       |
| Academic graphs                      | arXiv  | 1       | 169343*128 | 1166243(directed) | 40     |
| Commercial networks                  | Products | 1       | 2449029*100 | 123718280  | 47     |
| Biological networks                  | Proteins | 1       | 132,534    | 39561252*8 | 2       |
Figure A.4: The evaluation framework for small datasets

Figure A.5: A histogram of the number of nodes in each graph in the PATTERN and CLUSTER datasets

Appendix B.

GCN: Mathematically, the GCN model follows this formula:

\[ h^{(l+1)} = \sigma \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} h^{(l)} W^{(l)} \right) \]  

where \( H^{(l)} \) denotes the \( f^{(l)} \) layer in the network, \( \sigma \) is the nonlinearity, and \( W \) is the weight matrix for this layer. \( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \) indicates a renormalization trick in which there is a self-connection to each node of the graph. Therefore,
\( \hat{D} \) is the corresponding degree matrix of \( A + I \), and \( \tilde{A} = A + I \). The shape of \( H^{(0)} \) is \( N \times D \), where \( N \) is the number of nodes, and \( D \) is the number of input features. For better understanding, the following similar formula can also be used to describe the models.

\[
\begin{align*}
  h^{(l+1)}_i &= \sigma \left( \sum_{j \in N(i) \cup \{i\}} \frac{1}{\sqrt{\text{deg}(i)} \cdot \sqrt{\text{deg}(j)}} \left( h^{(l)}_j W^{(l)} \right) \right) \\
  \text{(B.2)}
\end{align*}
\]

where \( \sqrt{\text{deg}(i)} \) and \( \sqrt{\text{deg}(j)} \) are the degrees of nodes \( i \) and \( j \), respectively; \( N(i) \) is a neighbor of node \( i \); and \( \sigma \) is the activation function; here, we used ReLU as the activation function (see Figure B.6).

\[
\begin{align*}
  h^{(l)}_i &= \text{deg}^{0.5} \\
  \{h^{(l)}_j\} &= \text{deg}^{0.5} \\
  W^l &\rightarrow \text{ReLU} \\
  h^{(l+1)}_i &= h^{(l)}_i \\
  \text{Figure B.6: GCN Layer}
\end{align*}
\]

GraphSage: GraphSage acts as a framework for aggregating information about adjacent nodes. Under this framework, aggregate functions can be used to combine information from adjacent nodes. We then can access information from adjacent nodes when processing the current node. Finally, a norm is used for the vector that combines the information of the current node and adjacent nodes.

\[
\begin{align*}
  h^{(l+1)}_{N(i)} &= \text{aggregate} \left( \{h^{(l)}_j \mid j \in N(i)\} \right) \\
  h^{(l+1)}_i &= \sigma \left( \text{concat} \left( h^{(l)}_i, h^{(l+1)}_{N(i)} \right) \cdot W^l \right) \\
  h^{(l+1)}_i &= \text{norm} \left( h^{(l+1)}_i \right) \\
  \text{(B.3)} &\quad \text{(B.4)} &\quad \text{(B.5)}
\end{align*}
\]

where node \( j \) is a neighbor of \( i \), and \( \sigma \) is the activation function (see Figure B.7).

\[
\begin{align*}
  h^{(l)}_i &\rightarrow \{h^{(l)}_j\} \\
  W^l &\rightarrow \sigma \rightarrow \text{norm} \\
  h^{(l+1)}_i &\rightarrow h^{(l+1)}_i \\
  \text{Figure B.7: GraphSage Layer}
\end{align*}
\]

GatedGCN: In the formula, the number of input channels of \( x \) must be less than or equal to the number of output channels. If there are fewer input channels than output channels, we first use zero vectors to complete the input channels to obtain \( h^{(0)}_i \) to make the input channels equal to the output channels. \( \Theta \) is the learnable parameter, and \( e_{ij} \) is the edge weight. Finally, a gated recurrent unit (GRU) is used in the algorithm to implement long-term memory (see Figure B.8).

\[
\begin{align*}
  h^{(0)}_i &= x_i \parallel 0 \\
  m^{(l+1)}_i &= \sum_{j \in N(i)} \Theta \cdot e_{ij} \cdot h^{(l)}_j \\
  h^{(l+1)}_i &= \text{GRU} \left( m^{(l+1)}_i, h^{(l)}_i \right) \\
  \text{(B.6)} &\quad \text{(B.7)} &\quad \text{(B.8)}
\end{align*}
\]

Unlike the GCN, which adds the surrounding information, and GraphSAGE, which concatenates the surrounding information into its own embedding, GatedGCN uses the GRU method to collect the surrounding information and support modeling of long-term dependencies.
ResGatedGCN: In the ResGatedGCN formula, $A^l, B^l, C^l, D^l, E^l$ are all learnable parameters, $\varepsilon$ is a small fixed constant for numerical stability, and $\sigma$ is the sigmoid function. In detail, the edge gates can be regarded as a soft attention process, which is related to the standard sparse attention mechanism. In contrast to other GNNs, the GatedGCN architecture explicitly maintains the edge features $e_{ij}$ in each layer (see Figure B.9).

\[
\hat{e}_{ij}^l = E^l h_i^l + B^l h_j^l + C^l e_{ij}
\]

\[
\hat{e}_{ij}^{l+1} = ReLU(\hat{e}_{ij}^l)
\]

\[
h_i^{l+1} = ReLU\left(A^l h_i^l + \frac{\sum_{j \in N} \sigma(\hat{e}_{ij}) B^l h_j^l}{\sum_{j \in N} \sigma(\hat{e}_{ij}) + \varepsilon}\right)
\]

GAT: The core idea of the GAT is to learn the importance of neighboring nodes and then use the learned importance weight to carry out weighted summation to update a node’s own embedding.

\[
z_{ij}^{(l)} = W^{(l)} h_i^{(l)}
\]

\[
e_{ij}^{(l)} = \text{LeakyReLU}\left(\alpha_{ij}^{(l)} \left(z_{ij}^{(l)} | z_{ij}^{(l)}\right)\right)
\]

\[
\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in N_{ij}} \exp(e_{ik}^{(l)})}
\]
where \([\llbracket \cdot \rrbracket]\) indicates concatenating the features of node \(i\) and node \(j\), which have been added by the linear map of the shared parameter \(W\). \(\bar{d}^l(\cdot)\) is a learnable weight vector. A single-layer feedforward neural network is applied; then, LeakyReLU \((\cdot)\) and softmax \((\cdot)\) are used to calculate the attention coefficient \(\alpha_{ij}\). Finally, the node embedding is aggregated using the learned coefficient \(\alpha_{ij}\) to obtain the node embedding of the \(l + 1\) layer (see Figure B.10).

\[
\begin{align*}
h_i^{(l+1)} &= \sigma \left( \sum_{j \in \mathcal{N}(i) \cup \{i\}} a_{ij}^l \tilde{z}_j^l \right) \\
\end{align*}
\]  
(B.15)

Figure B.10: GAT layer

MONET: Mixture model networks are a deep network framework run in non-Euclidean space. In the case of graphs, the node update equation is defined as follows:

\[
\begin{align*}
z_i^{(l)} &= W^{(l)} h_i^{(l)} \\
h_i^{(l+1)} &= \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \sum_{k=1}^K w_k(e_{ij}) \odot \Theta_k h_j^l + h_i^{(l)} \Theta \\
w_k(e) &= \exp \left( -\frac{1}{2} (e - \mu_k)^T \Sigma_k^{-1} (e - \mu_k) \right) \\
e &= \tanh \left( \Theta \left( \text{deg}_{i}^{-1/2}, \text{deg}_{j}^{-1/2} \right)^T \right)
\end{align*}
\]  
(B.16) \hspace{1cm} (B.17) \hspace{1cm} (B.18) \hspace{1cm} (B.19)

where \(\mu_k\) and \(\Sigma_k^{-1}\) denote the (learnable) parameters of the mean vector and diagonal covariance matrix, respectively; \(\Theta_k\) and \(\Theta\) are the learnable parameters; and \(e\) and \(e_{ij}\) are the same. The edge attributes of nodes \(i\) and \(j\) are shown in Figure B.11.

Figure B.11: Monet layer
GIN: The equations below compute the node embedding $h_{i}^{(l+1)}$ of layer $l+1$ from the embeddings of layer $l$:

$$h_{i}^{(l+1)} = \text{MLP}^{(l)} \left( (1 + \epsilon^{(l)}) \cdot h_{i}^{(l)} + \sum_{j \in N(i)} h_{j}^{(l)} \right)$$  \hspace{1cm} (B.20)

where $l$ denotes the $l$-th layer in the network, $\epsilon$ is a learnable parameter or a fixed scalar, $N(i)$ denotes the set of neighbor indices of node $i$, and MLP represents the multilayer perceptron (see Figure B.12). In the first iteration, if the input features are one-hot encodings, MLP is not used before summation. For GIN, we consider the feature outputs from each layer of the network.

$$h_{i}^{l} \xrightarrow{1+\epsilon} \{h_{j}^{l}\} \xrightarrow{\text{sum}_j} \text{MLP} \xrightarrow{\text{h}_{i}^{l+1}}$$

![Figure B.12: GIN layer](image)

References

[1] W. L. Hamilton, R. Ying, J. Leskovec, Representation learning on graphs: Methods and applications, arXiv preprint arXiv:1709.05584 (2017).
[2] S. Bhagat, G. Cormode, S. Muthukrishnan, Node classification in social networks, in: Social Network Data Analytics, Springer, 2011, pp. 115–148.
[3] M. Zhang, Z. Cui, M. Neumann, Y. Chen, An end-to-end deep learning architecture for graph classification, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[4] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, J. Leskovec, Hierarchical graph representation learning with differentiable pooling, in: Advances in Neural Information Processing Systems, 2018, pp. 4800–4810.
[5] C. Cangea, P. Velickovic, N. Jovanovic, T. Kipf, P. Liò, Towards sparse hierarchical graph classifiers, arXiv preprint arXiv:1811.01287 (2018).
[6] F. M. Bianchi, D. Grattarola, C. Alippi, L. Livi, Graph neural networks with convolutional arma filters, arXiv preprint arXiv:1901.01343 (2019).
[7] L. C. Freeman, Visualizing social networks, Journal of social structure 1 (1) (2000) 4.
[8] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 701–710.
[9] D. Liben-Nowell, J. Kleinberg, The link-prediction problem for social networks, Journal of the American society for information science and technology 58 (7) (2007) 1019–1031.
[10] B. Yang, W.-t. Yih, X. He, J. Gao, L. Deng, Embedding entities and relations for learning and inference in knowledge bases, arXiv preprint arXiv:1412.6575 (2014).
[11] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).
[12] J. Zheng, Y. Wang, W. Xu, Z. Gan, P. Li, J. Lv, GSSA: Pay attention to graph feature importance for GCN via statistical self-attention, Neurocomputing 417 (2020) 458–470.
[13] G. Zhao, P. Jia, A. Zhou, B. Zhang, InfGCN: Identifying influential nodes in complex networks with graph convolutional networks, Neurocomputing 414 (2020) 18–26. doi:10.1016/j.neucom.2020.07.028
[14] G. Guo, W. Yuan, Short-term traffic speed forecasting based on graph attention temporal convolutional networks, Neurocomputing 410 (2020) 387–393. doi:10.1016/j.neucom.2020.06.001
[15] J. Wang, J. Hu, S. Qian, Q. Fang, C. Xu, Multimodal graph convolutional networks for high quality content recognition, Neurocomputing 412 (2020) 42–51. doi:10.1016/j.neucom.2020.04.145
[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2009, pp. 248–255.
[17] W. Hu, M. Fey, M. Mutnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec, Open graph benchmark: Datasets for machine learning on graphs, arXiv preprint arXiv:2005.00687 (2020).
[18] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, M. Li, Bag of tricks for image classification with convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 558–567.
[19] O. Shchur, M. Mumm, A. Bojchevski, S. Günnemann, Pitfalls of graph neural network evaluation, arXiv preprint arXiv:1811.05868 (2018).
[20] F. Errica, M. Podda, D. Bacciu, A. Micheli, A fair comparison of graph neural networks for graph classification, arXiv preprint arXiv:1912.09893 (2019).
[21] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S. Y. Philip, A comprehensive survey on graph neural networks, IEEE Transactions on Neural Networks and Learning Systems (2020).
[22] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, X. Bresson, Benchmarking graph neural networks, arXiv preprint arXiv:2003.00982 (2020).
[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8026–8037.
[24] M. Fey, J. E. Lenssen, Fast graph representation learning with PyTorch Geometric, arXiv preprint arXiv:1903.02428 (2019).
[25] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, C.-J. Hsieh, Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 257–266.
[26] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, V. Prasanna, Graphsaint: Graph sampling based inductive learning method, arXiv preprint arXiv:1907.04931 (2019).
[27] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, T. Eliassi-Rad, Collective classification in network data, AI magazine 29 (3) (2008) 93–93.
[28] Z. Yang, W. Cohen, R. Salakhudinov, Revisiting semi-supervised learning with graph embeddings, in: International Conference on Machine Learning, PMLR, 2016, pp. 40–48.
[29] E. Abbe, Community detection and stochastic block models: Recent developments, The Journal of Machine Learning Research 18 (1) (2017) 6446–6531.
[30] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (1) (2008) 61–80.
[31] K. Bhatia, K. Dahiyà, H. Jain, A. Mittal, Y. Prabhu, M. Varma, The extreme classification repository: Multi-label datasets and code (2016).
[32] K. Wang, Z. Shen, C. Huang, C.-H. Wu, Y. Dong, A. Kanakia, Microsoft academic graph: When experts are not enough, Quantitative Science Studies 1 (1) (2020) 396–413.
[33] D. Szklarczyk, A. L. Gable, D. Lyon, A. Junge, S. Wyder, J. Huerta-Cepas, M. Simonovic, N. T. Doncheva, J. H. Morris, P. Bork, STRING v11: Protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets, Nucleic acids research 47 (D1) (2019) D607–D613.
[34] G. O. Consortium, The gene ontology resource: 20 years and still GOing strong, Nucleic acids research 47 (D1) (2019) D330–D338.
[35] R. L. Murphy, B. Srinivasan, V. Rao, B. Ribeiro, Relational pooling for graph representations, arXiv preprint arXiv:1903.02541 (2019).
[36] B. Srinivasan, B. Ribeiro, On the equivalence between node embeddings and structural graph representations, arXiv preprint arXiv:1910.00452 (2019).
[37] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural computation 15 (6) (2003) 1373–1396.
[38] A. Grover, J. Leskovec, Node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864.
[39] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781 (2013).
[40] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, arXiv preprint arXiv:1710.10903 (2017).
[41] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in Neural Information Processing Systems, 2017, pp. 1024–1034.
[42] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, arXiv preprint arXiv:1810.00826 (2018).
[43] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, M. M. Bronstein, Geometric deep learning on graphs and manifolds using mixture model cnns, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5115–5124.
[44] X. Bresson, T. Laurent, Residual gated graph convnets, arXiv preprint arXiv:1711.07553 (2017).
[45] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, arXiv preprint arXiv:1511.05493 (2015).
[46] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
[47] Q. Li, Z. Han, X.-M. Wu, Deeper insights into graph convolutional networks for semi-supervised learning, arXiv preprint arXiv:1801.07606 (2018).
[48] Y. Rong, W. Huang, T. Xu, J. Huang, Dropedge: Towards deep graph convolutional networks on node classification, in: International Conference on Learning Representations, 2019.
[49] K. Zhou, Y. Dong, W. S. Lee, B. Hooi, H. Xu, J. Feng, Effective training strategies for deep graph neural networks, arXiv preprint arXiv:2006.07107 (2020).
[50] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R. P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: Advances in Neural Information Processing Systems, 2015, pp. 2224–2232.
[51] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, arXiv preprint arXiv:1409.0473 (2014).
[52] X. Bresson, T. Laurent, A two-step graph convolutional decoder for molecule generation, arXiv preprint arXiv:1906.03412 (2019).
[53] C. K. Joshi, T. Laurent, X. Bresson, An efficient graph convolutional network technique for the travelling salesman problem, arXiv preprint arXiv:1906.01227 (2019).