Benchmarking Graph Neural Networks

Vijay Prakash Dwivedi[1] Chaitanya K. Joshi[1] Thomas Laurent[2] Yoshua Bengio[3 4 5] Xavier Bresson[1]

Abstract

Graph neural networks (GNNs) have become the standard toolkit for analyzing and learning from data on graphs. They have been successfully applied to a myriad of domains including chemistry, physics, social sciences, knowledge graphs, recommendation, and neuroscience. As the field grows, it becomes critical to identify the architectures and key mechanisms which generalize across graphs sizes, enabling us to tackle larger, more complex datasets and domains. Unfortunately, it has been increasingly difficult to gauge the effectiveness of new GNNs and compare models in the absence of a standardized benchmark with consistent experimental settings and large datasets. In this paper, we propose a reproducible GNN benchmarking framework[6], with the facility for researchers to add new datasets and models conveniently. We apply this benchmarking framework to novel medium-scale graph datasets from mathematical modeling, computer vision, chemistry and combinatorial problems to establish key operations when designing effective GNNs. Precisely, graph convolutions, anisotropic diffusion, residual connections and normalization layers are universal building blocks for developing robust and scalable GNNs.

1. Introduction

Since the pioneering works of (Scarselli et al., 2009; Bruna et al., 2013; Defferrard et al., 2016; Sukhbaatar et al., 2016; Kipf & Welling, 2017; Hamilton et al., 2017), graph neural networks (GNNs) have seen a great surge of interest in recent years with promising methods being developed. As the field grows, the question on how to build powerful GNNs has become central. What types of architectures, first principles or mechanisms are universal, generalizable, and scalable to large datasets of graphs and large graphs? Another important question is how to study and quantify the impact of theoretical developments for GNNs? Benchmarking provides a strong paradigm to answer these fundamental questions. It has proved to be beneficial in several areas of science for driving progress, identifying essential ideas, and solving domain-specific problems (Weber et al., 2019). Recently, the famous 2012 ImageNet (Deng et al., 2009) challenge has provided a benchmark dataset that has triggered the deep learning revolution (Krizhevsky et al., 2012; Malik, 2017). International teams competed to produce the best predictive model for image classification on a large-scale dataset. Since breakthrough results on ImageNet, the Computer Vision community has forged the path forward towards identifying robust architectures and techniques for training deep neural networks (Zeiler & Fergus, 2014; Girshick et al., 2014; Long et al., 2015; He et al., 2016).

But designing successful benchmarks is highly challenging: it requires defining appropriate datasets, robust coding interfaces and common experimental setting for fair comparisons, all while being reproducible. Such requirements face several issues. First, how to define appropriate datasets? It may be hard to collect representative, realistic and large-scale datasets. This has been one of the most important issues with GNNs. Most published papers have been focused on quite small datasets like CORA and TU datasets (Kipf & Welling, 2017; Ying et al., 2018; Veličković et al., 2018; Xinyi & Chen, 2019; Xu et al., 2019; Lee et al., 2019), where all GNNs perform almost statistically the same. Somewhat counter-intuitively, baselines which do not consider graph structure perform equally well or, at times, better than GNNs (Errica et al., 2019). This has raised questions on the necessity of developing new and more complex GNN architectures, and even to the necessity of using GNNs (Chen et al., 2019). For example, in the recent works of Hoang & Maehara (2019) and Chen et al. (2019), the authors analyzed the capacity and components of GNNs to expose the limitations of the models on small datasets. They claim the datasets to be inappropriate for the design of complex structure-inductive learning architectures.

Another major issue in the GNN literature is to define com-

[1] School of Computer Science and Engineering, Nanyang Technological University, Singapore [2] Department of Mathematics, Loyola Marymount University [3] Mila [4] University of Montréal [5] CIFAR. Correspondence to: Vijay Prakash Dwivedi <vijaypra001@e.ntu.edu.sg>, Chaitanya K. Joshi <chaitanya.joshi@ntu.edu.sg>.

[6] https://github.com/graphdeeplearning/benchmarking-gnns

mon experimental settings. As noted in Errica et al. (2019), recent papers on TU datasets do not have a consensus on training, validation and test splits as well as evaluation protocols, making it unfair to compare the performance of new ideas and architectures. It is unclear how to perform good data splits beyond randomizes splits, which are known to provide over-optimistic predictions (Lohr, 2009). Additionally, different hyper-parameters, loss functions and learning rate schedules make it difficult to identify new advances in architectures.

This paper brings the following contributions:

• We release an open benchmark infrastructure for GNNs, hosted on GitHub based on PyTorch (Paszke et al., 2019) and DGL (Wang et al., 2019) libraries. We focus on ease-of-use for new users, making it easy to benchmark new datasets and GNN models.

• We aim to go beyond the popular but small CORA and TU datasets by introducing medium-scale datasets with 12k-70k graphs of variable sizes 9-500 nodes. Proposed datasets are from mathematical modeling (Stochastic Block Models), computer vision (super-pixels), combinatorial optimization (Traveling Salesman Problem) and chemistry (molecules' solubility).

• We identify important building blocks of GNNs with the proposed benchmark infrastructure. Graph convolutions, anistropic diffusion, residual connections, and normalization layers stick out as most useful to design efficient GNNs.

• We do not aim to rank published GNNs. It is computationally expensive (and beyond our resources) to find the best model for a specific task, as it would require an exhaustive search over hyper-parameter values with cross-validation. In contrast, we fix a parameter budget for all models and analyze performance trends to identify the important GNN mechanisms.

• The numerical results are entirely reproducible. We make it simple to reproduce the reported results by running scripts. Besides, the installation and execution of the benchmark infrastructure are explained in detail in the GitHub repository.

2. Proposed Benchmarking Framework

One of the goals of this work is to provide an easy to use collection of medium-scale datasets on which the different GNN architectures that have been proposed in the past few years exhibit clear and statistically meaningful differences in term of performance. We propose six datasets that are described in Table 1.

Table 1. Summary statistics of proposed benchmark datasets.

Domain/Construction	Dataset	# graphs	# nodes
Computer Vision/ Graphs constructed with super-pixels	MNIST CIFAR10	70K 60K	40-75 85-150
Chemistry/ Real-world molecular graphs	ZINC	12K	9-37
Artificial/ Graphs generated from Stochastic Block Model	PATTERN CLUSTER	14K 12K	50-180 40-190
Artificial/ Graphs generated from uniform distribution	TSP	12K	50-500

For the two computer vision datasets, each image from the classical MNIST (LeCun et al., 1998) and CIFAR10 (Krizhevsky et al., 2009) datasets were converted into graphs using so called super-pixels, see section 5.2. The task is then to classify these graphs into categories. The graphs in the PATTERN and CLUSTER datasets were generated according to a Stochastic Block Model, see section 5.4. The tasks consist of recognizing specific predetermined subgraphs (for the PATTERN dataset) or identifying clusters (for the CLUSTER dataset). These are node classification tasks. The TSP dataset is based on the Traveling Salesman Problem (Given a list of cities, what is the shortest possible route that visits each city and returns to the origin city?"), see section 5.5. We pose TSP on random Euclidean graphs as an edge classification/link prediction task, with the groundtruth value for each edge belonging to the TSP tour given by the Concorde solver (Applegate et al., 2006). ZINC, presented in section 5.3, is an already existing real-world molecular dataset. Each molecule can be converted into a graph: each atom becomes a node and each bond becomes an edge. The task is to regress a molecule property known as the constrained solubility (Jin et al., 2018).

Each of the proposed datasets contains at least $12,000$ graphs. This is in stark contrast with CORA and popularly used TU datasets, which often contain only a few hundreds of graphs. On the other hand, the proposed datasets are mostly artificial or semi-artificial (except for ZINC), which is not the case with the CORA and TU datasets. We therefore view these benchmarks as complementary to each other. The main motivation of our work is to propose datasets that are large enough so that differences observed between various GNN architecture are statistically relevant.

3. Graph Neural Networks

In their simplest form (Sukhbaatar et al., 2016; Kipf & Welling, 2017), graph neural networks iteratively update node representations from one layer to the other according to the formula:

$$\hat{h}_i^{\ell+1} = \frac{1}{\deg_i} \sum_{j \in \mathcal{N}_i} h_j^{\ell}, \qquad h_i^{\ell+1} = \sigma(U^{\ell}\, \hat{h}_i^{\ell+1}), \qquad (1)$$

where $h_i^{\ell+1}$ is the d-dimensional embedding representation of node i at layer $\ell+1$, $\mathcal{N}_i$ is the set of nodes connected to node i on the graph, $\deg_i = |\mathcal{N}_i|$ is the degree of node i, σ

is a nonlinearity, and $U^\ell \in \mathbb{R}^{d\times d}$ is a learnable parameter. We refer to this vanilla version of a graph neural network as GCN–Graph Convolutional Networks (Kipf & Welling, 2017). GraphSage (Hamilton et al., 2017) and GIN–Graph Isomorphism Network (Xu et al., 2019) propose simple variations of this averaging mechanism. In the mean version of GraphSage, the first equation of (1) is replaced with

$$\hat{h}_i^{\ell+1} = \text{Concat}\Big(h_i^\ell\ ,\ \frac{1}{\text{deg}_i}\sum_{j\in\mathcal{N}_i} h_j^\ell\Big), \tag{2}$$

and the embeddings vectors are projected onto the unit ball before being passed to the next layer. In the GIN architecture, the equations in (1) are replaced with

$$\hat{h}_i^{\ell+1} = (1+\epsilon)\, h_i^\ell + \sum_{j\in\mathcal{N}_i} h_j^\ell, \tag{3}$$

$$h_i^{\ell+1} = \sigma\Big(U^\ell\ \sigma\big(\ \text{BN}(\ V^\ell \hat{h}_i^{\ell+1}\ )\ \big)\Big), \tag{4}$$

where ϵ, U^ℓ, V^ℓ are learnable parameters and BN is the Batch Normalization layer (Ioffe & Szegedy, 2015). Importantly, GIN uses the features at all intermediate layers for the final prediction. In all the above models, each neighbor contributes equally to the update of the central node. We refer to these model as **isotropic**—they treat every “edge direction” equally.

On the other hand, MoNet–Gaussian Mixture Model Networks (Monti et al., 2017), GatedGCN–Graph Convolutional Networks (Bresson & Laurent, 2017), and GAT–Graph Attention Networks (Veličković et al., 2018) propose **anisotropic** update schemes of the type

$$\hat{h}_i^{\ell+1} = w_i^\ell h_i^\ell + \sum_{j\in\mathcal{N}_i} w_{ij}^\ell h_j^\ell, \tag{5}$$

where the weights w_i^ℓ and w_{ij}^ℓ are computed using various mechanisms (e.g. attention mechanism in GAT or gating mechanism in GatedGCN).

Finally, we also consider a hierarchical graph neural network, DiffPool–Differentiable Pooling (Ying et al., 2018), that uses the GraphSage formulation (2) at each stage of the hierarchy and for the pooling. Exact formulations for GNNs are available in the Supplementary Material. Refer to recent survey papers for a comprehensive overview of GNN literature (Bronstein et al., 2017; Zhou et al., 2018; Battaglia et al., 2018; Wu et al., 2019; Bacciu et al., 2019).

4. Issues with CORA and TU Datasets

The field of GNNs has mostly used the CORA and TU datasets. These datasets are realistic but they are also small. CORA has 2.7k nodes, TU-IMDB has 1.5k graphs with 13 nodes on average and TU-MUTAG has 188 molecules with 18 nodes. Although small datasets are useful to quickly develop new ideas[7], they can become a liability in the long run as new GNN models will be designed to overfit the small test sets, instead of searching for more generalizable architectures. CORA and TU datasets are examples of this overfitting problem.

As mentioned previously, another major issue with CORA and TU datasets is the lack of reproducibility of experimental results. Most published papers do not use the same train-validation-test split. Besides, even for the same split, the performances of GNNs present a large standard deviation on a regular 10-fold cross-validation because the datasets are too small. Our numerical experiments clearly show this, see section 5.1.

Errica et al. (2019) have recently introduced a rigorous evaluation framework to fairly compare 5 GNNs on 9 TU datasets for a single graph task–graph classification. This is motivated by earlier work by Shchur et al. (2018) on node classification, which highlighted GNN experimental pitfalls and the reproducibility issue. The paper by Errica et al. (2019) is an important first step towards a good benchmark. However, the authors only consider the small TU datasets and their rigorous evaluations are computationally expensive—they perform 47,000 experiments, where an experiment can last up to 48 hours. Additional tasks such as graph regression, node classification and edge classification are not considered, whereas the datasets are limited to the domains of chemistry and social networks. Open Graph Benchmark[8] is a recent initiative that is a very promising step toward the development of a benchmark of large real-world datasets from various domains.

5. Numerical Experiments

This section presents our numerical experiments with the proposed open-source benchmarking framework. Most GNN implementations, GCN–Graph Convolutional Networks (Kipf & Welling, 2017), GAT–Graph Attention Networks (Veličković et al., 2018), GraphSAGE (Hamilton et al., 2017), DiffPool–Differential Pooling (Ying et al., 2018), GIN–Graph Isomorphism Network (Xu et al., 2019), MoNet–Gaussian Mixture Model Networks (Monti et al., 2017), were taken from the Deep Graph Library (DGL) (Wang et al., 2019) and implemented in PyTorch (Paszke et al., 2019). We upgrade all DGL GNN implementations with residual connections (He et al., 2016), batch normalization (Ioffe & Szegedy, 2015) and graph size normalization. GatedGCN–Gated Graph Convolutional Networks (Bresson & Laurent, 2017) are the final GNN we consider,

[7] such as the older Caltech object recognition datasets, with a few hundred examples: `http://www.vision.caltech.edu/html-files/archive.html`

[8] `http://ogb.stanford.edu/`

with GatedGCN-E denoting the version which use edge attributes/features, if available in the dataset. Additionally, we implement a simple graph-agnostic baseline which parallel-ly applies an MLP on each node's feature vector, independent of other nodes. This is optionally followed by a gating mechanism to obtain the Gated MLP baseline (see Supplementary Material for details). We run experiments for TU, MNIST, CIFAR10, ZINC and TSP on Nvidia 1080Ti GPUs, and for PATTERN and CLUSTER on Nvidia 2080Ti GPUs.

5.1. Graph classification with TU Datasets

Our first experiment is graph classification on TU datasets[9]. We select three TU datasets—ENZYMES (480 train/60 validation/60 test graphs of sizes 2-126), DD (941 train/118 validation/119 test graphs of sizes 30-5748) and PROTEINS (889 train/112 validation/112 test graphs of sizes 4-620).

Here is the proposed benchmark protocol for TU datasets.
Splitting. We perform a 10-fold cross validation split which gives 10 sets of train, validation and test data indices in the ratio 8:1:1. We use stratified sampling to ensure that the class distribution remains the same across splits. The indices are saved and used across all experiments for fair comparisons.
Training. We use the Adam optimizer (Kingma & Ba, 2014) with a learning rate decay strategy. An initial learning rate is tuned from a range of $1e$–3 to $7e$–5 using grid search for every GNN models. The learning rate is reduced by half, *i.e.*, reduce factor 0.5, if the validation loss does not improve after 25 epochs. We do not set a maximum number of epochs—the training is stopped when the learning rate decays to a value of $1e$–6 or less. The model parameters at the end of training are used for evaluation on test sets.
Accuracy. We use classification accuracy between the predicted labels and groundtruth labels as our evaluation metric. Model performance is evaluated on the test split of the 10 folds for all TU datasets. The reported performance is the average and standard deviation over all the 10 folds.
Graph classifier layer. We use a graph classifier layer which first builds a graph representation by averaging all node features extracted from the last GNN layer and then passing this graph representation to a MLP.
Hyper-parameters and parameter budget. Our goal is not to find the optimal set of hyper-parameters for each dataset, but to identify performance trends. Thus, we fix a budget of around 100k parameters for all GNNs and arbitrarily select 4 layers. The number of hidden features is estimated to match the budget. An exception to this is DiffPool where we resort to the authors choice of using three graph convolutional layers each before and after the pooling layer. This selection is considered least to constitute a DiffPool based GNN model thus overshooting the learnable parameters above 100k for DD.

Table 2. Performance on the standard TU test sets (higher is better). Two runs of all the experiments over same hyperparameters but different random seeds are shown separately to note the differences in ranking and variation for reproducibility. The top 3 performance scores are highlighted as: **First**, **Second**, **Third**.

Dataset	Model	#Param	seed 1		seed 2	
			Acc ± s.d.	Epoch/Total	Acc ± s.d.	Epoch/Total
ENZYMES	MLP	62502	59.67±4.58	0.24s/0.22hr	54.33±4.90	0.26s/0.24hr
	MLP (Gated)	79014	62.50±4.10	0.20s/0.18hr	63.67±5.36	0.22s/0.20hr
	GCN	80038	63.50±4.44	0.83s/0.77hr	59.33±3.74	1.36s/1.19hr
	GraphSage	82686	**68.00±5.95**	0.90s/0.78hr	67.33±5.01	0.93s/0.84hr
	GIN	80770	**68.00±6.62**	0.59s/0.67hr	**68.17±5.84**	0.55s/0.61hr
	DiffPool	94782	65.33±2.96	2.05s/2.22hr	**67.50±5.74**	1.95s/2.01hr
	GAT	80550	66.33±5.52	6.69s/5.75hr	67.33±4.36	7.00s/5.93hr
	MoNet	83538	59.33±6.38	1.58s/1.46hr	57.50±5.28	1.74s/1.58hr
	GatedGCN	89366	**67.33±6.42**	2.31s/2.03hr	**68.00±4.14**	2.30s/2.00hr
DD	MLP	71458	72.24±3.43	1.17s/1.27hr	70.88±3.89	1.30s/1.34hr
	MLP (Gated)	87970	**78.53±2.80**	1.23s/1.25hr	77.85±2.90	1.02s/1.05hr
	GCN	88994	**77.84±2.27**	3.35s/2.37hr	**78.35±2.06**	4.51s/2.98hr
	GraphSage	89402	77.59±2.85	4.06s/2.58hr	**78.61±2.02**	4.67s/4.15hr
	GIN	85646	74.11±3.69	2.21s/1.80hr	73.52±3.93	2.02s/1.65hr
	DiffPool	165342	65.91±9.45	37.87s/33.42hr	63.73±1.49	37.48s/32.87hr
	GAT	89506	77.42±2.88	20.63s/11.86hr	**78.78±2.70**	23.20s/13.20hr
	MoNet	89134	77.08±2.71	25.39s/15.30hr	76.75±3.97	23.74s/14.92hr
	GatedGCN	98386	**78.35±1.74**	7.85s/7.20hr	78.10±1.93	8.44s/8.45hr
PROTEINS	MLP	63778	76.27±2.92	0.41s/0.29hr	**76.36±2.45**	0.35s/0.25hr
	MLP (Gated)	80290	75.10±3.85	0.36s/0.24hr	74.83±3.15	0.45s/0.28hr
	GCN	81314	**76.54±3.63**	1.74s/1.64hr	75.10±3.31	1.63s/1.55hr
	GraphSage	83642	76.18±4.14	1.72s/1.09hr	74.83±3.76	1.73s/1.22hr
	GIN	79886	69.62±5.13	1.04s/1.31hr	69.43±5.68	0.94s/1.25hr
	DiffPool	93780	**76.60±2.11**	3.99s/3.93hr	75.90±2.88	3.99s/3.90hr
	GAT	81826	75.55±3.09	11.55s/10.74hr	74.39±2.56	12.29s/11.40hr
	MoNet	84334	**77.26±3.12**	3.54s/2.99hr	**76.81±2.75**	3.22s/2.60hr
	GatedGCN	90706	76.45±3.77	3.58s/2.92hr	**76.00±2.19**	4.31s/2.96hr

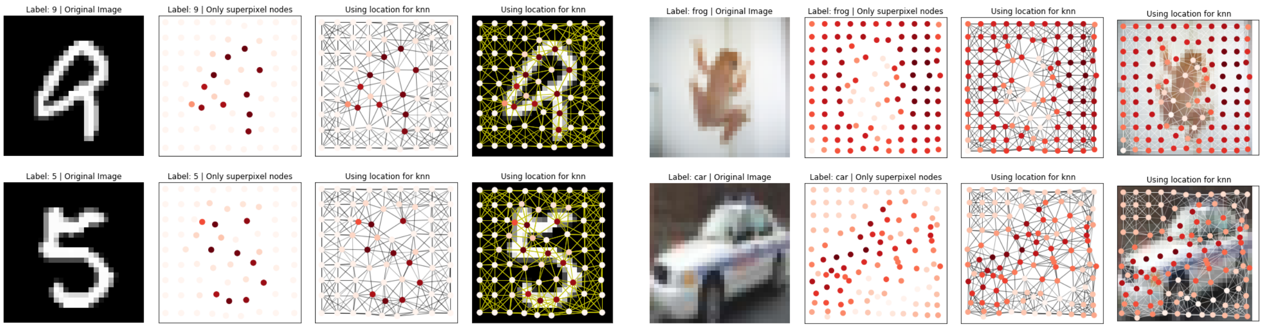
Our numerical results are presented in Table 2. All NNs have similar statistical test performance as the standard deviation is quite large. We also report a second run of these experiments with the same experimental protocol, *i.e.*, the same 10-fold splitting but different initialization. We observe a change of model ranking, which we attribute to the small size of the datasets and the non-determinism of gradient descent optimizers. We also observed that, for DD and PROTEINS, the graph-agnostic MLP baselines perform as good and sometimes better than GNNs.

5.2. Graph Classification with SuperPixel Datasets

For the second experiment, we use the popular MNIST and CIFAR10 image classification datasets from computer vision. The original MNIST and CIFAR10 images are converted to graphs using super-pixels. Super-pixels represent small regions of homogeneous intensity in images, and can be extracted with the SLIC technique (Achanta et al., 2012). We use SLIC super-pixels from (Knyazev et al., 2019)[10]. MNIST has 55000 train/5000 validation/10000 test graphs of sizes 40-75 nodes (*i.e.*, number of super-pixels) and CI-

[9] http://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

[10] https://github.com/bknyaz/graph_attention_pool


Label: 9 | Original Image
Label: 9 | Only superpixel nodes
Using location for knn
Using location for knn
Label: 5 | Original Image
Label: 5 | Only superpixel nodes
Using location for knn
Using location for knn
Label: frog | Original Image
Label: frog | Only superpixel nodes
Using location for knn
Using location for knn
Label: car | Original Image
Label: car | Only superpixel nodes
Using location for knn
Using location for knn

(a) MNIST

(b) CIFAR10

Figure 1. Sample images and their superpixel graphs. The graphs of SLIC superpixels (at most 75 nodes for MNIST and 150 nodes for CIFAR10) are 8-nearest neighbor graphs in the Euclidean space and node colors denote the mean pixel intensities.

FAR10 has 45000 train/5000 validation/10000 test graphs of sizes 85-150 nodes.

For each sample, we build a k-nearest neighbor adjacency matrix with $W_{ij} = \exp(-\|x_i - x_j\|^2/\sigma_x^2)$, where x_i, x_j are the 2-D coordinates of super-pixels i, j, and σ_x is the scale parameter defined as the averaged distance x_k of the k nearest neighbors for each node. Figure 1 presents visualizations of the super-pixel graphs of MNIST and CIFAR10.

We propose the following benchmark setup.
Splitting. We use the standard MNIST and CIFAR10 splits.
Training. We use the Adam optimizer with a learning rate decay strategy. For all GNNs, an initial learning rate is set to $1e$–3, the reduce factor is 0.5, the patience value is 5, and the stopping learning rate is $1e$–5.
Accuracy. The performance metric is the classification accuracy between predicted and groundtruth labels.
Reproducibility. We report accuracy averaged over 4 runs with 4 different random seeds. At each run, the same seed is used for all NNs.
Graph classifier layer. We use the same graph classifier as the TU dataset experiments, Section 5.1.
Hyper-parameters and parameter budget. We determine the model hyperparameters following the TU dataset experiments (Section 5.1) with a budget of 100k. Results for graph classification on MNIST and CIFAR10 are presented in Table 3 and analyzed in Section 6.

5.3. Graph Regression with Molecular Dataset

We use the ZINC molecular graphs dataset to regress a molecular property known as the constrained solubility (Jin et al., 2018). The statistics for ZINC are: 10000 train/1000 validation/1000 test graphs of sizes 9-37 nodes/atoms. For each molecular graph, node features are the type of atoms and edge features are the type of edges. The same experimental protocol as Section 5.2 is used, with the following changes:

Accuracy. The performance metric is the mean absolute error (MAE) between the predicted and the groundtruth constrained solubility.
Graph regression layer. The regression layer is similar to the graph classifier layer in section 5.2. Table 4 presents our numerical results, which we analyze in Section 6.

5.4. Node Classification with SBM Datasets

We consider the node-level tasks of graph pattern recognition (Scarselli et al., 2009) and semi-supervised graph clustering. The goal of graph pattern recognition is to find a fixed graph pattern P embedded in larger graphs G of variable sizes. Identifying patterns in different graphs is one of the most basic tasks for GNNs. The pattern and embedded graphs are generated with the stochastic block model (SBM) (Abbe, 2017). A SBM is a random graph which assigns communities to each node as follows: any two vertices are connected with the probability p if they belong to the same community, or they are connected with the probability q if they belong to different communities (the value of q acts as the noise level). For all experiments, we generate graphs G with 5 communities with sizes randomly generated between $[5, 35]$. The SBM of each community is $p = 0.5, q = 0.2$, and the signal on G is generated with a uniform random distribution with a vocabulary of size 3, *i.e.*, $\{0, 1, 2\}$. We randomly generate 100 patterns P composed of 20 nodes with intra-probability $p_P = 0.5$ and extra-probability $q_P = 0.5$ (*i.e.*, 50% of nodes in P are connected to G). The signal on P is also generated as a random signal with values $\{0, 1, 2\}$. The statistics for the PATTERN dataset are 10000 train/2000 validation/2000 test graphs of sizes 50-180 nodes. The output signal has value 1 if the node belongs to P and value 0 if it is in G.

The semi-supervised clustering task is another fundamental task in network science. We generate 6 SBM clusters with sizes randomly generated between $[5, 35]$ and probabil-

Table 3. Performance on the standard test sets of MNIST and CIFAR10 (higher is better). Results are averaged over 4 runs with 4 different seeds. **Red**: the best model, Violet: good models. **Bold** indicates the best model between residual and non-residual connections (both models are bold if they perform equally).

Dataset	Model	#Param	Residual		No Residual	
			Acc	Epoch/Total	Acc	Epoch/Total
MNIST	MLP	104044	*not used*		94.46±0.28	21.82s/1.02hr
	MLP (Gated)	105717	*not used*		95.18±0.18	22.43s/0.73hr
	GCN	101365	**89.99±0.15**	78.25s/1.81hr	**89.05±0.21**	79.18s/1.76hr
	GraphSage	102691	**97.09±0.02**	75.57s/1.36hr	**97.20±0.17**	76.80s/1.42hr
	GIN	105434	**93.91±0.63**	34.30s/0.73hr	**93.96±1.30**	34.61s/0.74hr
	DiffPool	106538	**95.02±0.42**	170.55s/4.26hr	**94.66±0.48**	171.38s/4.45hr
	GAT	110400	**95.62±0.13**	375.71s/6.35hr	**95.56±0.16**	377.06s/6.35hr
	MoNet	104049	**90.36±0.47**	581.86s/15.31hr	**89.73±0.48**	567.12s/12.05hr
	GatedGCN	104217	**97.37±0.06**	128.39s/2.01hr	**97.36±0.12**	127.15s/2.13hr
	GatedGCN-E*	104217	**97.24±0.10**	135.10s/2.25hr	**97.47±0.13**	127.86s/2.15hr
CIFAR10	MLP	104044	*not used*		56.01±0.90	21.82s/1.02hr
	MLP (Gated)	106017	*not used*		56.78±0.12	27.85s/0.68hr
	GCN	101657	**54.46±0.10**	100.91s/2.73hr	51.64±0.45	100.30s/2.44hr
	GraphSage	102907	**65.93±0.30**	96.67s/1.88hr	**66.08±0.24**	96.00s/1.79hr
	GIN	105654	**53.28±3.70**	45.29s/1.24hr	47.66±0.47	44.30s/0.93hr
	DiffPool	108042	**57.99±0.45**	298.06s/10.17hr	56.84±0.37	299.64s/10.42hr
	GAT	110704	**65.40±0.38**	389.40s/7.32hr	**65.48±0.33**	386.14s/7.75hr
	MoNet	104229	**53.42±0.43**	836.32s/22.45hr	50.99±0.17	869.90s/21.79hr
	GatedGCN	104357	**69.19±0.28**	146.80s/2.48hr	**68.92±0.38**	145.14s/2.49hr
	GatedGCN-E*	104357	68.64±0.60	158.80s/2.74hr	**69.37±0.48**	145.66s/2.43hr

*GatedGCN-E uses the graph adjacency weight as edge feature.

ities $p = 0.55, q = 0.25$. The statistics for the CLUSTER dataset is 10000 train/1000 validatin/1000 test graphs of sizes 40-190 nodes. We only provide a single label randomly selected for each community. The output signal is defined as the cluster class label. We follow the same experimental protocol as Section 5.2, with the following changes:
Accuracy. The performance metric is the average accuracy over the classes.
Node classification layer. For classifying each node, we pass the node features from the last GNN layer to a MLP. Results for node classification on CLUSTER and PATTERN are presented in Table 5 and analyzed in Section 6.

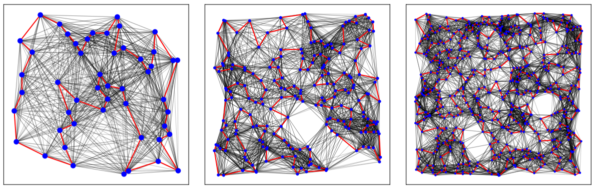
5.5. Edge Classification with TSP Dataset

Leveraging machine learning for solving NP-hard combinatorial optimization problems (COPs) has been the focus of intense research in recent years (Vinyals et al., 2015; Bengio et al., 2018). Recently proposed deep learning-based solvers for COPs (Khalil et al., 2017; Li et al., 2018; Kool et al., 2019) combine GNNs with classical graph search to predict approximate solutions directly from problem instances (represented as graphs). Consider the intensively studied Travelling Salesman Problem (TSP): given a 2D Euclidean graph, one needs to find an optimal sequence of nodes, called a tour, with minimal total edge weights (tour length). TSP's *multi-scale* nature makes it a challenging graph task which requires reasoning about both local node neighborhoods as well as global graph structure.

Table 4. Performance on the standard test sets of ZINC (lower is better). Results are averaged over 4 runs with 4 different seeds. **Red**: the best model and **Violet**: good models. **Bold** indicates the best model between residual and non-residual connections (both models are bold if they perform equally).

Model	#Param	Residual		No Residual	
		Acc/MAE	Epoch/Total	Acc/MAE	Epoch/Total
MLP	108975	*not used*		0.710±0.001	1.19s/0.02hr
MLP (Gated)	106970	*not used*		0.681±0.005	1.16s/0.03hr
GCN	103077	**0.469±0.002**	3.02s/0.08hr	0.525±0.007	2.97s/0.09hr
GraphSage	105031	0.429±0.005	3.24s/0.10hr	**0.410±0.005**	3.20s/0.10hr
GIN	103079	**0.414±0.009**	2.49s/0.06hr	**0.408±0.008**	2.50s/0.06hr
DiffPool	110561	**0.466±0.006**	12.41s/0.34hr	0.514±0.007	12.36s/0.38hr
GAT	102385	**0.463±0.002**	20.97s/0.56hr	0.496±0.004	21.03s/0.62hr
MoNet	106002	**0.407±0.007**	11.69s/0.28hr	0.444±0.024	11.75s/0.34hr
GatedGCN	105735	0.437±0.008	6.36s/0.17hr	**0.422±0.006**	6.12s/0.17hr
GatedGCN-E*	105875	**0.363±0.009**	6.34s/0.17hr	**0.365±0.009**	6.17s/0.17hr

*GatedGCN-E uses the molecule bond type as edge feature.



(a) TSP50 (b) TSP200 (c) TSP500

Figure 2. Sample graphs from the TSP dataset. Nodes are colored blue and edges on the groundtruth TSP tours are colored red.

For our experiments with TSP, we follow the learning-based approach to COPs described in (Li et al., 2018; Joshi et al., 2019), where a GNN is the backbone architecture for assigning probabilities to each edge as belonging/not belonging to the predicted solution set. The probabilities are then converted into discrete decisions through graph search techniques. We create train, validation and test sets of 10000, 1000 and 1000 TSP instances, respectively, where each instance is a graph of n node locations sampled uniformly in the unit square $S = \{x_i\}_{i=1}^n$ and $x_i \in [0, 1]^2$. We generate problems of varying size and complexity by uniformly sampling the number of nodes $n \in [50, 500]$ for each instance.

In order to isolate the impact of the backbone GNN architectures from the search component, we pose TSP as a binary edge classification task, with the groundtruth value for each edge belonging to the TSP tour given by Concorde. For scaling to large instances, we use sparse $k = 25$ nearest neighbor graphs instead of full graphs, following Khalil et al. (2017). See Figure 2 for sample TSP instances of various sizes. We follow the same experimental protocol as Section 5.2 with the following changes:
Training. The patience value is 10 by default. For additional experiments on the impact of model depth, we use a patience value of 5.
Accuracy. Given the high class imbalance, *i.e.*, only the edges in the TSP tour have positive label, we use the F1 score for the positive class as our performance metric.
Reproducibility. We report F1 scores averaged over 2 runs with 2 different random seeds.

Table 5. Performance on the standard test sets of PATTERN and CLUSTER SBM graphs (higher is better). Results are averaged over 4 runs with 4 different seeds. **Red**: the best model and Violet: good models. **Bold** indicates the best model between residual and non-residual connections.

Dataset	Model	#Param	Residual		No Residual	
			Acc	Epoch/Total	Acc	Epoch/Total
PATTERN	MLP	105263	*not used*		50.13±0.00	8.68s/0.10hr
	MLP (Gated)	103629	*not used*		50.13±0.00	9.78s/0.12hr
	GCN	100923	**74.36±1.59**	97.37s/2.06hr	55.22±0.17	97.46s/2.30hr
	GraphSage	98607	78.20±3.06	79.19s/2.57hr	**81.25±3.84**	79.43s/2.14hr
	GIN	100884	96.98±2.18	14.12s/0.32hr	**98.25±0.38**	14.11s/0.37hr
	GAT	109936	**90.72±2.04**	229.76s/5.73hr	88.91±4.48	229.65s/8.78hr
	MoNet	103775	95.52±3.74	879.87s/21.80hr	**97.89±0.89**	870.05s/24.86hr
	GatedGCN	104003	95.05±2.80	115.55s/2.46hr	**97.24±1.19**	115.03s/2.59hr
CLUSTER	MLP	106015	*not used*		20.97±0.01	6.54s/0.08hr
	MLP (Gated)	104305	*not used*		20.97±0.01	7.37s/0.09hr
	GCN	101655	**47.82±4.91**	66.58s/1.26hr	34.85±0.65	66.81s/1.21hr
	GraphSage	99139	44.89±3.70	54.53s/1.05hr	**53.90±4.12**	54.40s/1.19hr
	GIN	103544	49.64±2.09	11.60s/0.27hr	**52.54±1.03**	11.57s/0.27hr
	GAT	110700	49.08±6.47	158.23s/4.08hr	**54.12±1.21**	158.46s/4.53hr
	MoNet	104227	**45.95±3.39**	635.77s/15.32hr	39.48±2.21	600.04s/11.18hr
	GatedGCN	104355	**54.20±3.58**	81.39s/2.26hr	50.18±3.03	80.66s/2.07hr

Edge classifier layer. To make a prediction for each edge e_{ij}, we first concatenate node features h_i and h_j from the final GNN layer. The concatenated features are then passed to an MLP for prediction.

Non-learnt Baseline. In addition to reporting performance of GNNs, we compare with a simple k-nearest neighbor heuristic baseline, defined as follows: Predict true for the edges corresponding to the k nearest neighbors of each node, and false for all other edges. We set $k = 2$ for optimal performance. Comparing GNNs to the non-learnt baseline tells us whether models learn something more sophisticated than identifying a node's nearest neighbors. Our numerical results are presented in Tables 6 and analyzed in Section 6.

6. What did we learn?

Graph-agnostic NNs (MLP) perform as well as GNNs on small datasets. Tables 2 and 3 show there is no significant improvement by using GNNs over graph-agnostic MLP baselines for the small TU datasets and the (simple) MNIST. Besides, MLP can sometimes do better than GNNs (Errica et al., 2019; Luzhnica et al., 2019), such as for the DD dataset.

GNNs improve upon graph-agnostic NNs for larger datasets. Tables 4 and 5 present a significant gain of performance for the ZINC, PATTERN, and CLUSTER datasets, in which all GNNs vastly outperform the two MLP baselines. Table 6 shows that all GNNs using residual connections surpass the MLP baselines for the TSP dataset. Results reported in Table 3 for the CIFAR10 dataset are less discriminative, although the best GNNs perform notably better than MLPs.

Table 6. Performance on TSP test set graphs with and without residual connections (higher is better). Results are averaged over 2 runs with 2 different seeds. **Red**: the best model and Violet: good models. **Bold** indicates the best model between residual and non-residual connections (both models are bold if they perform equally).

Model	#Param	Residual		No Residual	
		F1	Epoch/Total	F1	Epoch/Total
k-NN Heuristic	k=2	F1: 0.693			
MLP	94394	*not used*		0.548±0.003	53.92s/2.85hr
MLP (Gated)	115274	*not used*		0.548±0.001	54.39s/2.44hr
GCN	108738	**0.627±0.003**	163.36s/11.26hr	0.547±0.003	164.41s/10.28hr
GraphSage	98450	**0.663±0.003**	145.75s/16.05hr	**0.657±0.002**	147.22s/14.33hr
GIN	118574	**0.655±0.001**	73.09s/5.44hr	**0.657±0.001**	74.71s/5.60h
GAT	109250	**0.669±0.001**	360.92s/30.38hr	0.567±0.003	360.74s/20.55hr
MoNet	94274	**0.637±0.010**	1433.97s/41.69hr	0.569±0.002	1472.65s/42.44hr
GatedGCN	94946	**0.794±0.004**	203.28s/15.47hr	**0.791±0.003**	202.12s/15.20hr
GatedGCN-E*	94946	**0.802±0.001**	201.40s/15.19hr	0.794±0.003	201.32s/15.05hr

*GatedGCN-E uses the pairwise distance as edge feature.

***Vanilla* GCNs (Kipf & Welling, 2017) have poor performance.** GCNs are the simplest form of GNNs. Their node representation update relies on an isotropic averaging operation over the neighborhood, Eq.(1). This isotropic property was analyzed in (Chen et al., 2019) and was shown to be unable to distinguish simple graph structures, explaining the low performance of GCNs across all datasets.

New isotropic GNN architectures improve on GCN. GraphSage (Hamilton et al., 2017) demonstrates the importance of using the central node information in the graph convolutional layer, Eq.(2). GIN (Xu et al., 2019) also employs the central node feature, Eq.(3), along with a new classifier layer that connects to convolutional features at all intermediate layers. DiffPool (Ying et al., 2018) considers a learnable graph pooling operation where GraphSage is used at each resolution level. These three isotropic GNNs significantly improve the performance of GCN for all datasets, apart from CLUSTER.

Anisotropic GNNs are accurate. Anisotropic models such as GAT (Veličković et al., 2018), MoNet (Monti et al., 2017) and GatedGCN (Bresson & Laurent, 2017) obtain the best results for each dataset, with the exception of PATTERN. Also, we note that GatedGCN performs consistently well across all datasets. Unlike isotropic GNNs that mostly rely on a simple sum over the neighboring features, anisotropic GNNs employ complex mechanisms (sparse attention mechanism for GAT, edge gates for GatedGCN) which are harder to implement efficiently. Our code for these models is not fully optimized and, as a result, much slower.

An additional advantage of this class of GNNs is their ability to explicitly use edge features, such as the bond type between two atoms in a molecule. In Table 4, for the ZINC molecular dataset, GatedGCN-E using the bond edge features significantly improved the MAE performance of GatedGCN without bonds.

Residual connections improve performance. Residual

Table 7. Performance on TSP test set graphs for deep GNNs (up to 32 layers) with and without residual connections (higher is better). L denotes number of layers. **Bold** indicates the best model between residual and non-residual connections (both models are bold if they perform equally).

Model	L	**#Param**	**Residual**		**No Residual**	
			F1	**Epoch/Total**	**F1**	**Epoch/Total**
GCN	4	108738	**0.628**	165.15s/5.69hr	0.552	168.72s/6.14hr
	8	175810	**0.639**	279.33s/9.86hr	0.568	281.56s/14.16hr
	16	309954	**0.651**	502.59s/21.37hr	0.532	507.35s/10.72hr
	32	578242	**0.666**	1042.46s/28.96hr	0.361	1031.62s/15.19hr
GIN	4	118574	**0.653**	71.41s/2.50hr	**0.653**	75.63s/3.34hr
	8	223866	**0.675**	93.95s/4.26hr	0.674	93.41s/5.19hr
	16	434450	**0.681**	146.09s/5.68hr	0.642	144.52s/2.89hr
	32	855618	**0.669**	274.5s/3.81hr	0.6063	282.97s/4.40hr
GatedGCN	4	94946	**0.792**	214.67s/6.50hr	**0.787**	212.67s/9.75hr
	8	179170	**0.817**	374.39s/14.56hr	0.807	367.68s/19.72hr
	16	347618	**0.833**	685.41s/22.85hr	0.810	678.76s/22.07hr
	32	684514	**0.843**	1760.56s/48.00hr	0.722	1760.55s/33.27hr

connections (RC), introduced in (He et al., 2016), have become a universal ingredient in deep learning architectures for computer vision. Using residual links helps GNNs in two ways. First, it limits the vanishing gradient problem during backpropagation in deep networks. Second, it allows the inclusion of self-node information during convolution in models like GCN and GAT, which do not use them explicitly.

We first test the influence of RC with $L = 4$ layers. For MNIST (Table 3), RC do not improve the performance as most GNNs are able to easily overfit this dataset. For CIFAR10, RC enhance results for GCN, GIN, DiffPool and MoNet, but they do not help or degrade the performance for GraphSAGE, GAT and GatedGCN. For ZINC (Table 4), adding residuality significantly improves GCN, DiffPool, GAT, and MoNet, but it slightly degrades the performance of GIN, GraphSage and GatedGCN. For PATTERN and CLUSTER (Table 5), GCN is the only architecture that clearly benefits from RC, while the other models can see their accuracy increase or decrease in the presence of RC. For TSP (Table 6), models which do not implicitly use self-information (GCN, GAT, MoNet) benefit from skip connections while other GNNs hold almost the same performance.

Next, we evaluate the impact of RC for deep GNNs. Figure 3 and Table 7 present the results of deep GNNs for ZINC, CLUSTER and TSP with $L = 4, 8, 16, 32$ layers. Interestingly, all models benefit from residual links when the number of layers increase, expect GIN, which is already equipped with skip connections for readout—the classification layer is always connected to all intermediate convolutional layers. In summary, our results suggest residual connections are an important building block for designing deep GNNs.

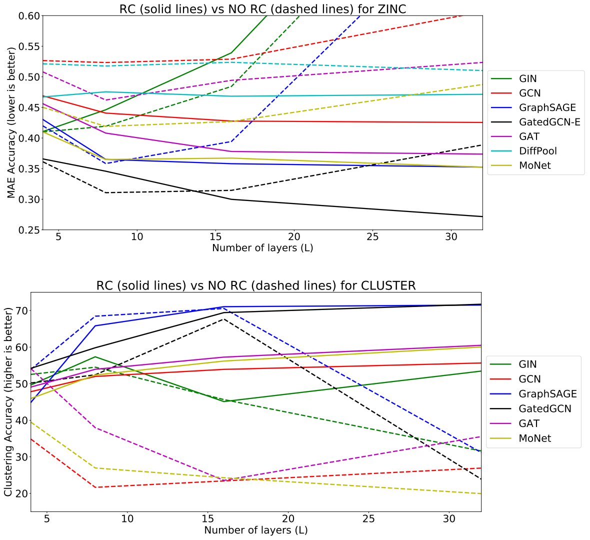


Figure 3. Performance on ZINC and CLUSTER test set graphs for deep GNNs up to 32 layers with (solid lines) and without (dashed lines) residual connections. Results are averaged over 4 runs with 4 different seeds.

Normalization layers can improve learning. Most real-world graph datasets are collections of irregular graphs with varying graph sizes. Batching graphs of variable sizes may lead to node representation at different scales. Hence, normalizing activations can be helpful to improve learning and generalization. We use two normalization layers in our experiments—batch normalization (BN) from (Ioffe & Szegedy, 2015) and graph size normalization (GN). Graph size normalization is a simple operation where the resulting node features h_i are normalized w.r.t. the graph size, *i.e.*, $h_i^\ell \leftarrow h_i^\ell \times \frac{1}{\sqrt{\mathcal{V}}}$, where $\mathcal{V}$ is the number of nodes. This normalization layer is applied after the convolutional layer and before the activation layer.

We evaluate the impact of the normalization layers on ZINC, CIFAR10 and CLUSTER datasets in Table 8. For the three datasets, BN and GN significantly improve GAT and GatedGCN. Besides, BN and GN boost GCN performance for ZINC and CLUSTER, but do not improve for CIFAR10. GraphSage and DiffPool do not benefit from normalizations but do not lose performance, except for CLUSTER. Additionally, GIN slightly benefits from normalization for ZINC and CLUSTER, but degrades for CIFAR10. We perform an ablation study in the Supplementary Material to study the influence of each normalization layer. In summary, normalization layers can be critical to design sound GNNs.

7. Conclusion

In this paper, we propose a benchmarking framework to facilitate the study of graph neural networks, and address experimental inconsistencies in the literature. We confirm how the widely used small TU datasets are inappropriate to

Table 8. Performance on ZINC, CIFAR10 and CLUSTER test set graphs with and without BN (batch normalization) and GN (graph normalization). Results are averaged over 4 runs with 4 different seeds and shown as **Acc ± s.d.** (lower is better for ZINC and higher is better for CIFAR10 and CLUSTER). **Bold** indicates the best model between using and not using the normalization layers (both models are bold if they perform equally).

Dataset	Model	#Param	BN & GN	No BN & GN
ZINC	MLP	108975	*not used*	0.710±0.001
	MLP (Gated)	106970	*not used*	0.683±0.004
	GCN	103077	**0.469±0.002**	0.490±0.007
	GraphSage	105031	**0.429±0.005**	**0.431±0.005**
	GIN	103079	**0.414±0.009**	0.426±0.010
	DiffPool	110561	**0.466±0.006**	**0.465±0.008**
	GAT	102385	**0.463±0.002**	0.487±0.006
	MoNet	106002	**0.407±0.007**	0.477±0.009
	GatedGCN-E	105875	**0.363±0.009**	0.399±0.003
CIFAR10	MLP	104044	*not used*	56.01±0.90
	MLP (Gated)	106017	*not used*	56.78±0.12
	GCN	101657	**54.46±0.10**	**54.14±0.67**
	GraphSage	102907	**65.93±0.30**	**65.98±0.15**
	GIN	105654	53.28±3.70	**55.49±1.54**
	DiffPool	108042	**57.99±0.45**	**56.70±0.71**
	GAT	110704	**65.40±0.38**	62.72±0.36
	GatedGCN-E	104357	**68.64±0.60**	64.10±0.44
CLUSTER	MLP	106015	*not used*	20.97±0.01
	MLP (Gated)	104305	*not used*	20.97±0.01
	GCN	101655	**47.82±4.91**	27.05±5.79
	GraphSage	99139	44.89±3.70	**48.83±3.84**
	GIN	103544	**49.64±2.09**	47.60±1.05
	GAT	110700	**49.08±6.47**	40.04±4.90
	GatedGCN	104355	**54.20±3.58**	34.05±2.57

examine innovations in this field and introduce six medium-scale datasets within the framework. Our experiments on multiple tasks for graphs show: i) Graph structure is important as we move towards larger datasets; ii) GCN, the simplest isotropic version of GNNs, cannot learn complex graph structures; iii) Self-node information, hierarchy, attention mechanisms, edge gates and better readout functions are key structures to improve GCN; iv) GNNs can scale deeper using residual connections and performance can be improved using normalization layers. As a final note, our benchmarking infrastructure, leveraging PyTorch and DGL, is fully reproducible and open to users on GitHub to experiment with new models and add datasets.

Acknowledgement

XB is supported by NRF Fellowship NRFF2017-10.

References

Abbe, E. Community Detection and Stochastic Block Models: Recent Developments. *arXiv preprint arXiv:1703.10146*, 2017.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Susstrunk, S. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):22742282, November 2012. ISSN 0162-8828.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. Concorde tsp solver, 2006.

Bacciu, D., Errica, F., Micheli, A., and Podda, M. A gentle introduction to deep learning for graphs. *arXiv preprint arXiv:1912.12693*, 2019.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pp. 4502–4510, 2016.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*, 2018.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Bresson, X. and Laurent, T. A two-step graph convolutional decoder for molecule generation. In *NeurIPS Workshop on Machine Learning and the Physical Sciences*, 2019.

Brockschmidt, M. Gnn-film: Graph neural networks with feature-wise linear modulation. *arXiv preprint arXiv:1906.12192*, 2019.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 1842, Jul 2017. ISSN 1053-5888.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Chen, T., Bian, S., and Sun, Y. Are powerful graph neural nets necessary? a dissection on graph classification, 2019.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29*, pp. 3844–3852. 2016.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Errica, F., Podda, M., Bacciu, D., and Micheli, A. A fair comparison of graph neural networks for graph classification, 2019.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Hoang, N. and Maehara, T. Revisiting graph neural networks: All we have is low-pass filters. *ArXiv*, abs/1905.09550, 2019.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Jin, W., Barzilay, R., and Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

Joshi, C. K., Laurent, T., and Bresson, X. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6348–6358, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Knyazev, B., Taylor, G. W., and Amer, M. R. Understanding attention and generalization in graph neural networks. *arXiv preprint arXiv:1905.02850*, 2019.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012.*, pp. 1106–1114, 2012.

Krizhevsky, A. et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lee, J., Lee, I., and Kang, J. Self-attention graph pooling. In *Proceedings of the 36th International Conference on Machine Learning*, 09–15 Jun 2019.

Li, Z., Chen, Q., and Koltun, V. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pp. 539–548, 2018.

Lohr, S. L. *Sampling: design and analysis*. Nelson Education, 2009.

Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Luzhnica, E., Day, B., and Liò, P. On graph classification networks, datasets and baselines. *arXiv preprint arXiv:1905.04682*, 2019.

Malik, J. Technical perspective: What led computer vision to deep learning? *Commun. ACM*, 60(6):8283, May 2017. ISSN 0001-0782.

Marcheggiani, D. and Titov, I. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*, 2017.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. doi: 10.1109/cvpr.2017.576.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pp. 4470–4479, 2018.

Scarselli, F., Gori, M., Tsoi, A., Hagenbuchner, M., and Monfardini, G. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Shchur, O., Mumme, M., Bojchevski, A., and Gnnemann, S. Pitfalls of graph neural network evaluation, 2018.

Sukhbaatar, S., szlam, a., and Fergus, R. Learning multiagent communication with backpropagation. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2244–2252. 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *Advances in Neural Information Processing Systems*, pp. 2692–2700, 2015.

Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., Huang, Z., Guo, Q., Zhang, H., Lin, H., Zhao, J., Li, J., Smola, A. J., and Zhang, Z. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Weber, L. M., Saelens, W., Cannoodt, R., Soneson, C., Hapfelmeier, A., Gardner, P. P., Boulesteix, A.-L., Saeys, Y., and Robinson, M. D. Essential guidelines for computational method benchmarking. *Genome biology*, 20(1): 125, 2019.

Weisfeiler, B. and Lehman, A. A. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9): 12–16, 1968.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks, 2019.

Xinyi, Z. and Chen, L. Capsule graph neural network. In *International Conference on Learning Representations*, 2019.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems 31*, pp. 4800–4810. 2018.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications, 2018.

A. Formalism

This section formally describes our experimental pipeline, illustrated in Figure 5. We detail the components of the setup including the input layers, the GNN layers and the MLP prediction layers.

A.1. Input layer

Given a graph, we are given node features $\alpha_i \in \mathbb{R}^{a\times 1}$ for each node i and (optionally) edge features $\beta_{ij} \in \mathbb{R}^{b\times 1}$ for each edge connecting node i and node j. The input features α_i and β_{ij} are embedded to d-dimensional hidden features $h_i^{\ell=0}$ and $e_{ij}^{\ell=0}$ via a simple linear projection before passing them to a graph neural network:

$$h_i^0 = U^0\alpha_i + u^0 \;\; ; \;\; e_{ij}^0 = V^0\beta_{ij} + v^0, \tag{6}$$

where $U^0 \in \mathbb{R}^{d\times a}$, $V^0 \in \mathbb{R}^{d\times b}$ and $u^0, v^0 \in \mathbb{R}^d$. If the input node/edge features are one-hot vectors of discrete variables, then biases u^0, v^0 are not used.

A.2. Graph neural network layers

Each GNN layer computes d-dimensional representations for the nodes/edges of the graph through recursive neighborhood diffusion (or message passing), where each graph node gathers features from its neighbors to represent local graph structure. Stacking L GNN layers allows the network to build node representations from the L-hop neighborhood of each node.

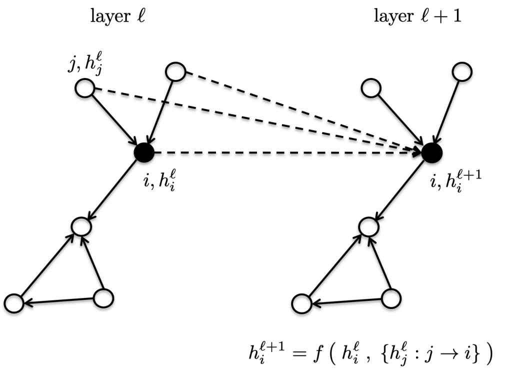


Figure 4. A generic graph neural network layer. Figure adapted from Bresson & Laurent (2017).

Let h_i^ℓ denote the feature vector at layer ℓ associated with node i. The updated features $h_i^{\ell+1}$ at the next layer $\ell+1$ are obtained by applying non-linear transformations to the central feature vector h_i^ℓ and the feature vectors h_j^ℓ for all nodes j in the neighborhood of node i (defined by the graph structure). This guarantees the transformation to build local reception fields, such as in standard ConvNets for computer vision, and be invariant to both graph size and vertex re-indexing.

Table 9. Characteristics of the benchmarked GNN variants.

Model	Self info.	Graph info.	Aggregation	Anisotropy	Heads/Kernels
MLP	✓	x	-	-	-
GCN	x	✓	Mean	x	Single
GraphSage	✓	✓	Mean, Max, LSTM	x	Single
GIN	✓	✓	Mean, Max, Sum	x	Single
GAT	x	✓	Weighted mean	✓(Attention)	Multi-head
MoNet	x	✓	Weighted sum	✓(Pseudo-edges)	Multi-kernel
GatedGCN	✓	✓	Weighted mean	✓(Edge gates)	Single

Thus, the most generic version of a feature vector $h_i^{\ell+1}$ at vertex i at the next layer in the graph network is:

$$h_i^{\ell+1} = f\,\left(\,h_i^\ell\,,\,\{h_j^\ell : j \to i\}\,\right), \tag{7}$$

where $\{j \to i\}$ denotes the set of neighboring nodes j pointed to node i, which can be replaced by $\{j \in \mathcal{N}_i\}$, the set of neighbors of node i, if the graph is undirected. In other words, a GNN is defined by a mapping f taking as input a vector h_i^ℓ (the feature vector of the center vertex) as well as an un-ordered set of vectors $\{h_j^\ell\}$ (the feature vectors of all neighboring vertices), see Figure 4. The arbitrary choice of the mapping f defines an instantiation of a class of GNNs. See Table 9 for an overview of the GNNs we study in this paper.

***Vanilla* Graph ConvNets (GCN) (Kipf & Welling, 2017)** In the simplest formulation of GNNs, Graph ConvNets iteratively update node features via an isotropic averaging operation over the neighborhood node features, *i.e.*,

$$h_i^{\ell+1} = \text{ReLU}\Big(U^\ell \,\text{Mean}_{j\in\mathcal{N}_i}\, h_j^\ell\Big), \tag{8}$$

$$= \text{ReLU}\Big(U^\ell \,\frac{1}{\text{deg}_i}\sum_{j\in\mathcal{N}_i} h_j^\ell\Big), \tag{9}$$

where $U^\ell \in \mathbb{R}^{d\times d}$ (a bias is also used, but omitted for clarity purpose), deg_i is the in-degree of node i, see Figure 6. Eq. (8) is called a *convolution* as it is a linear approximation of a localized spectral convolution. Note that it is possible to add the central node features h_i^ℓ in the update (8) by using self-loops or residual connections, see Section D.

GraphSage (Hamilton et al., 2017) GraphSage improves upon the simple GCN model by explicitly incorporating each node's own features from the previous layer in its update equation:

$$\hat{h}_i^{\ell+1} = \text{ReLU}\Big(U^\ell \,\text{Concat}\,\big(h_i^\ell,\, \text{Mean}_{j\in\mathcal{N}_i}\, h_j^\ell\big)\Big), \tag{10}$$

where $U^\ell \in \mathbb{R}^{d\times 2d}$, see Figure 7. Observe that the transformation applied to the central node features h_i^ℓ is different to

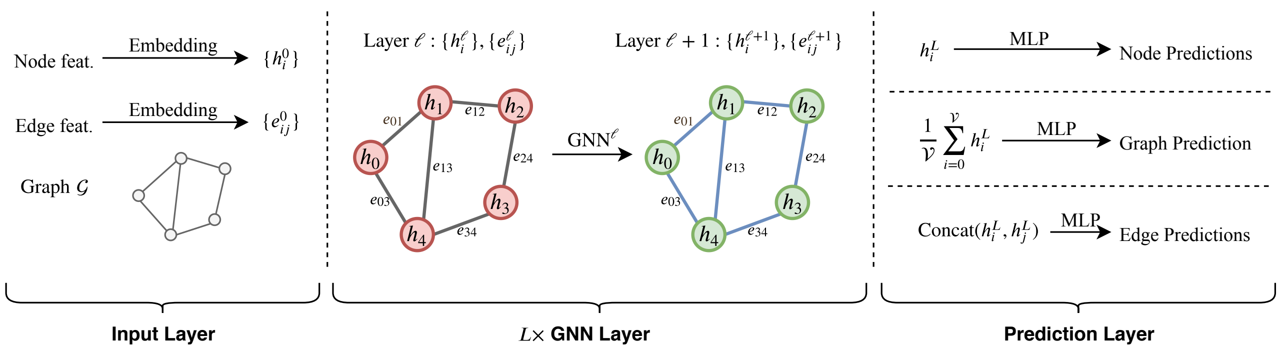


Figure 5. A standard experimental pipeline, which embeds the graph node and edge features, performs several GNN layers to compute convolutional features, and finally makes a prediction through a task-specific MLP layer.

the transformation carried out to the neighborhood features h_j^ℓ. The node features are then projected onto the ℓ_2-unit ball before being passed to the next layer:

$$h_i^{\ell+1} = \frac{\hat{h}_i^{\ell+1}}{\|\hat{h}_i^{\ell+1}\|_2}. \tag{11}$$

The authors also define more sophisticated neighborhood aggregation functions, such as Max-pooling or LSTM aggregators:

$$\hat{h}_i^{\ell+1} = \text{ReLU}\Big(U^\ell \text{ Concat}\left(h_i^\ell,\ \text{Max}_{j\in\mathcal{N}_i} V^\ell h_j^\ell\right)\Big), \tag{12}$$

$$\hat{h}_i^{\ell+1} = \text{ReLU}\Big(U^\ell \text{ Concat}\Big(h_i^\ell,\ \text{LSTM}^\ell_{j\in\mathcal{N}_i}\left(h_j^\ell\right)\Big)\Big), \tag{13}$$

where $V^\ell \in \mathbb{R}^{d\times d}$ and the LSTM$^\ell$ cell also uses learnable weights. In our experiments, we use the mean version of GraphSage, Eq.(10) (numerical experiments with the max version did not show significant differences, see Section E.2).

Graph Isomorphism Networks (GIN) (Xu et al., 2019) The GIN architecture is based the Weisfeiler-Lehman Isomorphism Test (Weisfeiler & Lehman, 1968) to study the expressive power of GNNs. The node update equation is defined as:

$$\begin{aligned} h_i^{\ell+1} &= \text{ReLU}\big(\,U^\ell\big(\text{ReLU}\big(\,\text{BN}\big(\,V^\ell \hat{h}_i^{\ell+1}\big)\,\big)\,\big)\,\big), && (14)\\ \hat{h}_i^{\ell+1} &= (1+\epsilon)\,h_i^\ell + \sum_{j\in\mathcal{N}_i} h_j^\ell, && (15)\end{aligned}$$

where ϵ is a learnable constant, $U^\ell, V^\ell \in \mathbb{R}^{d\times d}$, BN denoted Batch Normalization (described in subsequent sections), see Figure 8.

Graph Attention Network (GAT) (Veličković et al., 2018) GAT uses the attention mechanism of (Bahdanau et al., 2014) to introduce anisotropy in the neighborhood aggregation function. The network employs a multi-headed architecture to increase the learning capacity, similar to the Transformer (Vaswani et al., 2017). The node update equation is given by:

$$h_i^{\ell+1} = \text{Concat}_{k=1}^K\Big(\text{ELU}\Big(\sum_{j\in\mathcal{N}_i} e_{ij}^{k,\ell}\ U^{k,\ell}\ h_j^\ell\Big)\Big), \tag{16}$$

where $U^{k,\ell} \in \mathbb{R}^{\frac{d}{K}\times d}$ are the K linear projection heads, and $e_{ij}^{k,\ell}$ are the attention coefficients for each head defined as:

$$e_{ij}^{k,\ell} = \frac{\exp(\hat{e}_{ij}^{k,\ell})}{\sum_{j'\in\mathcal{N}_i}\exp(\hat{e}_{ij'}^{k,\ell})}, \tag{17}$$

$$\hat{e}_{ij}^{k,\ell} = \text{LeakyReLU}\Big(V^{k,\ell}\ \text{Concat}\big(U^{k,\ell}h_i^\ell,\ U^{k,\ell}h_j^\ell\big)\Big), \tag{18}$$

where $V^{k,\ell} \in \mathbb{R}^{\frac{2d}{K}}$, see Figure 9. GAT learns a mean over each node's neighborhood features sparsely weighted by the importance of each neighbor.

MoNet (Monti et al., 2017) The MoNet model introduces a general architecture to learn on graphs and manifolds using the Bayesian Gaussian Mixture Model (GMM) (Dempster et al., 1977). In the case of graphs, the node update equation is defined as:

$$\begin{aligned} h_i^{\ell+1} &= \text{ReLU}\Big(\sum_{k=1}^K\sum_{j\in\mathcal{N}_i} e_{ij}^{k,\ell}\ U^{k,\ell}\ h_j^\ell\Big), && (19)\\ e_{ij}^{k,\ell} &= \exp\Big(-\frac{1}{2}(u_{ij}^\ell-\mu_k^\ell)^T(\Sigma_k^\ell)^{-1}(u_{ij}^\ell-\mu_k^\ell)\Big), && (20)\\ u_{ij}^\ell &= \text{Tanh}\Big(A^\ell(\text{deg}_i^{-1/2},\text{deg}_j^{-1/2})^T + a^\ell\Big), && (21)\end{aligned}$$

where $U^{k,\ell} \in \mathbb{R}^{d\times d}$, $\mu_k^\ell, \left(\Sigma_k^\ell\right)^{-1}, a^\ell \in \mathbb{R}^2$ and $A^\ell \in \mathbb{R}^{2\times 2}$ are the (learnable) parameters of the GMM, see Figure 10.

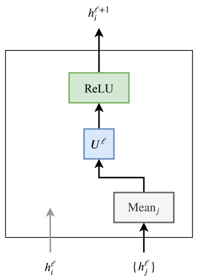


Figure 6. GCN Layer

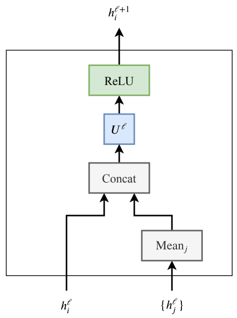


Figure 7. GraphSage Layer

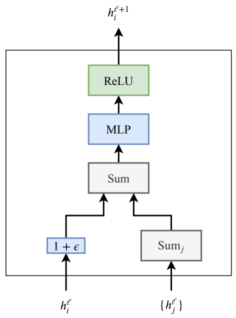


Figure 8. GIN Layer

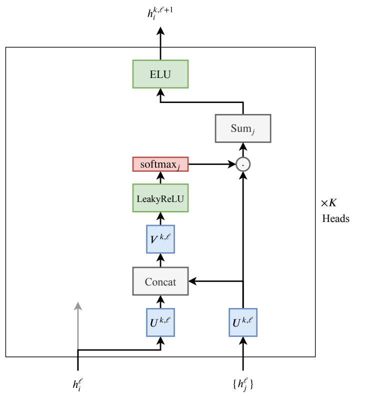


Figure 9. GAT Layer

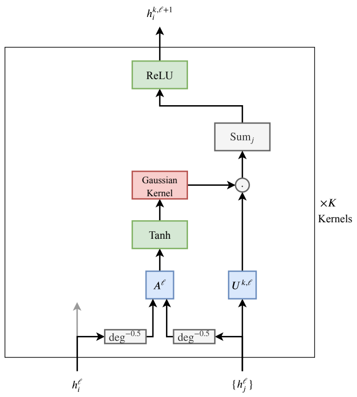


Figure 10. MoNet Layer

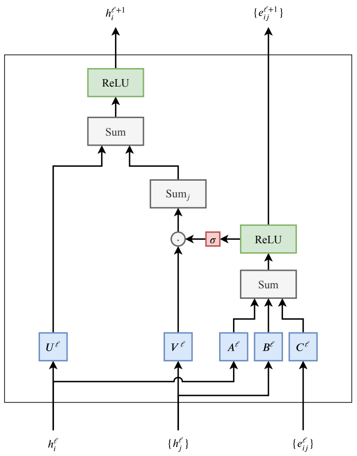


Figure 11. GatedGCN Layer

Gated Graph ConvNet (GatedGCN) (Bresson & Laurent, 2017) GatedGCN considers residual connections, batch normalization and edge gates to design another anisotropic variant of GCN. The authors propose to explicitly update edge features along with node features:

$$h_i^{\ell+1} = h_i^\ell + \text{ReLU}\Big(\text{BN}\Big(U^\ell h_i^\ell + \sum_{j\in\mathcal{N}_i} e_{ij}^\ell \odot V^\ell h_j^\ell\Big)\Big), \quad (22)$$

where $U^\ell, V^\ell \in \mathbb{R}^{d\times d}$, $\odot$ is the Hadamard product, and the edge gates e_{ij}^ℓ are defined as:

$$e_{ij}^\ell = \frac{\sigma(\hat{e}_{ij}^\ell)}{\sum_{j'\in\mathcal{N}_i}\sigma(\hat{e}_{ij'}^\ell)+\varepsilon}, \quad (23)$$

$$\hat{e}_{ij}^\ell = \hat{e}_{ij}^{\ell-1} + \text{ReLU}\Big(\text{BN}\big(A^\ell h_i^{\ell-1} + B^\ell h_j^{\ell-1} + C^\ell \hat{e}_{ij}^{\ell-1}\big)\Big), (24)$$

where σ is the sigmoid function, ε is a small fixed constant for numerical stability, $A^\ell, B^\ell, C^\ell \in \mathbb{R}^{d\times d}$, see Figure 11. Note that the edge gates (23) can be regarded as a soft attention process, related to the standard sparse attention mechanism (Bahdanau et al., 2014). Different from other anisotropic GNNs, the GatedGCN architecture explicitly maintains edge features $\hat{e}_{ij}$ at each layer, following (Bresson & Laurent, 2019; Joshi et al., 2019).

Note on anisotropic GNNs Anisotropic GNNs, such as MoNet, GAT and GatedGCN, generally update node features as:

$$\hat{h}_i^{\ell+1} = w_i^\ell h_i^\ell + \sum_{j\in\mathcal{N}_i} w_{ij}^\ell \, h_j^\ell. \quad (25)$$

Eq. (25) can be regarded as a learnable non-linear anisotropic diffusion operator on graphs where the discrete diffusion time is defined by ℓ, the index of the layer. Anisotropy does not come naturally on graphs. As arbitrary graphs have no specific orientations (like up, down, left, right directions in an image), a diffusion process on graphs is consequently *isotropic*, making all neighbors equally important. However, this may not be true in general, *e.g.*, a neighbor in the same community of a node shares different information than a neighbor in a separate community. GAT makes the diffusion process *anisotropic* with the attention mechanism (Bahdanau et al., 2014; Vaswani et al., 2017). MoNet uses the node degrees as edge features, and GatedGCN employs learneable edge gates such as in Marcheggiani & Titov (2017). We believe the anisotropic property to be critical in designing GNNs. Anisotropic models learn the best edge representations for encoding special information flow on the graph structure for the task to be solved.

Other examples of anisotropic GNN architectures include Interaction Networks (Battaglia et al., 2016; Sanchez-Gonzalez et al., 2018) and GNN-FiLM (Brockschmidt, 2019).

A.3. Normalization and Residual Connections

Irrespective of the class of GNNs used, we augment each GNN layer with batch normalization (BN) (Ioffe & Szegedy, 2015), graph size normalization (GN) and residual connections (He et al., 2016). As such, we consider a more specific class of GNNs than (7):

$$h_i^{\ell+1} = h_i^{\ell} \; + \, \sigma\Big(\text{BN}\left(\text{GN}\left(\hat{h}_i^{\ell+1}\right)\right)\Big), \tag{26}$$

$$\hat{h}_i^{\ell+1} = g_{\text{GNN}}\left(\, h_i^{\ell} \,,\, \{h_j^{\ell} : j \to i\} \,\right), \tag{27}$$

where σ is a non-linear activation function and g_{GNN} is a specific GNN layer.

Batch Normalization (Ioffe & Szegedy, 2015) As a reminder, BatchNorm normalizes each mini-batch of m feature vectors using the mini-batch mean and variance, as follows:

$$\mu^{\ell} = \frac{1}{m}\sum_{i=0}^{m} h_i^{\ell} \;\; ; \;\; \sigma^{\ell} = \sqrt{\frac{1}{m}\sum_{i=0}^{m}\left(h_i^{\ell} - \mu^{\ell}\right)}, \tag{28}$$

and then replace each h_i^{ℓ} with its normalized version, followed by a learnable affine transformation:

$$\hat{h}_i^{\ell} = W_{\text{BN}}\left(\frac{h_i^{\ell} - \mu^{\ell}}{\sigma^{\ell}}\right) + b_{\text{BN}}, \tag{29}$$

where $W_{\text{BN}}, b_{\text{BN}} \in \mathbb{R}^{d\times d}$.

Graph Size Normalization Batching graphs of variable sizes may lead to node representation at different scales, making it difficult to learn the optimal statistics μ and σ for BatchNorm. Therefore, we consider a GraphNorm layer to normalize the node features h_i^{ℓ} w.r.t. the graph size, *i.e.*,

$$\bar{h}_i^{\ell} = h_i^{\ell} \times \frac{1}{\sqrt{\mathcal{V}}}, \tag{30}$$

where $\mathcal{V}$ is the number of graph nodes. The GraphNorm layer is placed before the BatchNorm layer.

A.4. Non-graph MLP baselines (MLP, MLP-Gated)

In addition to benchmarking various classes of GNNs, we consider a simple baseline using graph-agnostic networks for obtaining node features. We apply a MLP on each nodes feature vector, independently of other nodes, *i.e.*,

$$h_i^{\ell+1} = \text{ReLU}\Big(U^{\ell} h_i^{\ell}\Big), \tag{31}$$

where $U^{\ell} \in \mathbb{R}^{d\times d}, \ell = 1, ..., L$. This defines our MLP layer. We also consider a slight upgrade by using a gating mechanism, which (independently) scales each node's final layer features through a sigmoid function:

$$h_i^{L} = \sigma\left(V h_i^{L-1}\right) \cdot \text{ReLU}\left(U^{L-1} h_i^{L-1}\right), \tag{32}$$

where $V \in \mathbb{R}^{d\times d}, \sigma$ is the sigmoid function. This establishes our second baseline, called MLP-Gated.

A.5. Prediction layers

The final component of each network is a prediction layer to compute task-dependent outputs, which will be given to a loss function to train the network parameters in an end-to-end manner. The input of the prediction layer is the result of final the GNN layer for each node of the graph (except GIN, which uses features from all intermediate layers).

Graph classifier layer To perform graph classification, we first build a d-dimensional graph-level vector representation $y_{\mathcal{G}}$ by averaging over all node features in the final GNN layer:

$$y_{\mathcal{G}} = \frac{1}{\mathcal{V}}\sum_{i=0}^{\mathcal{V}} h_i^{L}, \tag{33}$$

The graph features are then passed to a MLP, which outputs un-normalized logits/scores $y_{\text{pred}} \in \mathbb{R}^{C}$ for each class:

$$y_{\text{pred}} = P\,\text{ReLU}\left(Q\, y_{\mathcal{G}}\right), \tag{34}$$

where $P \in \mathbb{R}^{d\times C}, Q \in \mathbb{R}^{d\times d}, C$ is the number of classes. Finally, we minimize the cross-entropy loss between the logits and groundtruth labels.

Graph regression layer For graph regression, we compute $y_{\mathcal{G}}$ using Eq.(33) and pass it to a MLP which gives the prediction score $y_{\text{pred}} \in \mathbb{R}$:

$$y_{\text{pred}} = P\,\text{ReLU}\left(Q\, y_{\mathcal{G}}\right), \tag{35}$$

where $P \in \mathbb{R}^{d\times 1}, Q \in \mathbb{R}^{d\times d}$. The L1-loss between the predicted score and the groundtruth score is minimized during the training.

Table 10. Summary statistics of all datasets. Numbers in parentheses of Node features and Edge features are the dimensions.

Dataset	#Graphs	#Classes	Avg. Nodes	Avg. Edges	Node feat. (dim)	Edge feat. (dim)
ENZYMES	600	6	32.63	62.14	Node Attr (18)	N.A.
DD	1178	2	284.32	715.66	Node Label (89)	N.A.
PROTEINS	1113	2	39.06	72.82	Node Attr (29)	N.A.
MNIST	70000	10	70.57	564.53	Pixel+Coord (3)	Node Dist (1)
CIFAR10	60000	10	117.63	941.07	Pixel[RGB]+Coord (5)	Node Dist (1)
ZINC	12000	–	23.16	49.83	Node Label (28)	Edge Label (4)
PATTERN	14000	2	117.47	4749.15	Node Attr (3)	N.A.
CLUSTER	12000	6	117.20	4301.72	Node Attr (7)	N.A.
TSP	12000	2	275.76	6894.04	Coord (2)	Node Dist (1)

Node classifier layer For node classification, we independently pass each node's feature vector to a MLP for computing the un-normalized logits $y_{i,\text{pred}} \in \mathbb{R}^C$ for each class:

$$y_{i,\text{pred}} = P\,\text{ReLU}\left(Q\,h_i^L\right), \tag{36}$$

where $P \in \mathbb{R}^{d\times C}, Q \in \mathbb{R}^{d\times d}$. The cross-entropy loss weighted inversely by the class size is used during training.

Edge classifier layer To make a prediction for each graph edge e_{ij}, we first concatenate node features h_i and h_j from the final GNN layer. The concatenated edge features are then passed to a MLP for computing the un-normalized logits $y_{ij,\text{pred}} \in \mathbb{R}^C$ for each class:

$$y_{ij,\text{pred}} = P\,\text{ReLU}\left(Q\,\text{Concat}\left(h_i^L,\,h_j^L\right)\right), \tag{37}$$

where $P \in \mathbb{R}^{d\times C}, Q \in \mathbb{R}^{d\times 2d}$. The standard cross-entropy loss between the logits and groundtruth labels is used.

B. Dataset Statistics

Summary statistics for various datasets used in this paper are shown in Table 10. Histograms of nodes and edges are shown in Figure 13.

C. Normalization Layers

In this section, we perform an ablation study on Batch Normalization (BN) and Graph Size Normalization (GN) to empirically study the impact of normalization layers in GNNs. The results, drawn from graph regression (ZINC), graph classification (CIFAR10) and node classification (CLUSTER), are summarized in Table 11. We draw the following inferences.

GCN, GAT and GatedGCN. For these GNNs, GN followed by BN consistently improves performances. This empirically demonstrates the necessity to normalize the learnt features w.r.t. activation ranges and graph sizes for better training and generalization. Importantly, GN must always complement BN and is not useful on its own—solely using GN, in the absence of BN, hurts performance across all datasets compared to using no normalization.

Table 11. Performance on ZINC, CIFAR10 and CLUSTER test set graphs with and without BN (batch normalization) and GN (graph normalization). Results are averaged over 4 runs with 4 different seeds and shown as **Acc ± s.d.** (lower is better for ZINC and higher is better for CIFAR10 and CLUSTER). **Bold** indicates the best models that are statistically close.

Dataset	Model	#Param	BN & GN	BN & NO_GN	NO_BN & GN	NO_BN & NO_GN
ZINC	MLP	108975	*not used*	*not used*	*not used*	0.710±0.001
	MLP (Gated)	106970	*not used*	*not used*	*not used*	0.683±0.004
	GCN	103077	**0.469±0.002**	0.486±0.006	0.537±0.005	0.490±0.007
	GraphSage	105031	**0.429±0.005**	**0.428±0.004**	**0.424±0.008**	**0.431±0.005**
	GIN	103079	**0.414±0.009**	**0.407±0.011**	0.456±0.009	0.426±0.010
	DiffPool	110561	**0.466±0.006**	**0.469±0.006**	**0.470±0.004**	**0.465±0.008**
	GAT	102385	**0.463±0.002**	0.479±0.004	0.509±0.008	0.487±0.006
	MoNet	106002	**0.407±0.007**	0.418±0.008	0.455±0.010	0.477±0.009
	GatedGCN-E	105875	**0.363±0.009**	0.394±0.003	0.389±0.004	0.399±0.003
CIFAR10	MLP	104044	*not used*	*not used*	*not used*	56.01±0.90
	MLP (Gated)	106017	*not used*	*not used*	*not used*	56.78±0.12
	GCN	101657	**54.46±0.10**	**54.92±0.08**	49.27±0.97	**54.14±0.67**
	GraphSage	102907	**65.93±0.30**	**66.02±0.19**	**65.96±0.26**	**65.98±0.15**
	GIN	105654	53.28±3.70	51.07±2.17	**59.09±2.24**	55.49±1.54
	DiffPool	108042	**57.99±0.45**	**57.18±1.01**	**57.25±0.29**	**56.70±0.71**
	GAT	110704	**65.40±0.38**	**65.45±0.28**	60.19±0.82	62.72±0.36
	GatedGCN-E	104357	**68.64±0.60**	**69.74±0.35**	59.57±1.30	64.10±0.44
CLUSTER	MLP	106015	*not used*	*not used*	*not used*	20.97±0.01
	MLP (Gated)	104305	*not used*	*not used*	*not used*	20.97±0.01
	GCN	101655	**47.82±4.91**	39.76±6.90	21.00±0.04	27.05±5.79
	GraphSage	99139	44.89±3.70	45.58±5.65	46.27±6.61	**48.83±3.84**
	GIN	103544	**49.64±2.09**	**48.50±2.39**	44.11±4.10	47.60±1.05
	GAT	110700	**49.08±6.47**	43.32±8.03	30.20±7.82	40.04±4.90
	GatedGCN	104355	**54.20±3.58**	50.38±5.02	27.80±3.32	34.05±2.57

GraphSage and GIN. Using BN and GN neither improves nor degrades the performance of GraphSage (except on CLUSTER). Intuitively, the implicit ℓ_2 normalization in GraphSage, Eq.(11), acts similar to BN, controlling the range of activations.

Numerical results for GIN are not consistent across the datasets. For ZINC and CLUSTER, BN is useful with or without using GN. However, the sole usage of GN offers the best result. We hypothesize this discrepancy to be caused by the conflict between the internal BN in the GIN layer, Eq.(14), and the BN we placed generically after each graph convolutional layer, Eq.(26).

Table 12. Performance of GCN on test set graphs with and without SL (self-loops) and RC (residual connections). Performance is measured as **Acc** ± **s.d.** (lower is better for ZINC and higher is better for other datasets). Results are averaged over 4 runs with 4 different seeds. **Bold** indicates the best models that are statistically close.

Dataset	#Param	NO_SL & RC		NO_SL & NO_RC		SL & RC		SL & NO_RC	
		Acc	Epoch/Total	Acc	Epoch/Total	Acc	Epoch/Total	Acc	Epoch/Total
ZINC	103077	**0.469±0.002**	3.02s/0.08hr	**0.469±0.005**	3.00s/0.08hr	0.482±0.010	3.18s/0.08hr	0.491±0.005	3.13s/0.09hr
MNIST	101365	**89.99±0.15**	78.25s/1.81hr	89.29±0.36	79.41s/1.85hr	**89.72±0.28**	86.78s/2.11hr	89.29±0.36	79.41s/1.85hr
CIFAR10	101657	**54.46±0.10**	100.91s/2.73hr	51.40±0.17	100.37s/2.67hr	**53.87±0.93**	113.43s/3.23hr	52.03±0.35	113.01s/2.60hr
PATTERN	100923	**74.36±1.59**	97.37s/2.06hr	55.37±0.22	97.60s/2.56hr	**74.57±1.39**	102.01s/2.56hr	54.94±0.43	101.69s/2.38hr
CLUSTER	101655	**47.82±4.91**	66.58s/1.26hr	36.23±0.68	66.85s/1.28hr	41.95±7.14	70.55s/1.42hr	40.87±0.79	70.37s/1.27hr

D. Self-Loops and Residual Connections

The GCN model, Eq.(8), is not explicitly designed to include the central node feature h_i^ℓ in the computation of the next layer feature $h_i^{\ell+1}$. To solve this issue, the authors decided to augment the graphs with self-loops (SL). An alternative solution to the self-loop trick is the use of residual connections (RC). RC allow to explicitly include the central node features in the network architecture without increasing the size of the graphs. Besides, RC are particularly effective when designing deep networks as they limit the vanishing gradient problem. This led us to carry out an ablation study on self-loops and residual connections. The results are presented in Table 12.

RC without SL gives the best performance. Overall, the best results are obtained with RC in the absence of SL. Similar performances are obtained for MNIST, CIFAR10 and PATTERN when using RC and SL. However, the use of SL increases the size of the graphs and, consequently, the computational time (up to 12%).

Decoupling central node features from neighborhood features is critical for node-level tasks. It is interesting to notice the significant gain in accuracy for node classification tasks (PATTERN and CLUSTER) when using RC and NO SL vs. using SL and NO RC.

When using SL and NO RC, the same linear transformation is applied to the central node and the neighborhood nodes. For RC and NO SL, the central vertex is treated differently from the neighborhood vertices. Such distinction is essential to treat each node to be class-wise distinct. For graph-level tasks like graph regression (ZINC) or graph classification (MNIST and CIFAR10), the final graph representation is the mean of the node features, therefore showing a comparatively lesser margin of gain.

E. Additional Experiments on TSP

E.1. Why is GatedGCN suitable for edge tasks?

The TSP edge classification task, although artificially generated, presents an interesting empirical result. With the exception of GatedGCN, no GNN is able to outperform the non-learnt k-nearest neighbor heuristic baseline (F1 score: 0.69). This lead us to further explore the suitability of GatedGCN architecture for edge classification, see Table 13.

Impact of scale. We study how the GatedGCN architecture scales, training models with as few as $3,610$ parameters up to $684,514$ parameters. Naturally, performance improves with larger models and more layers. Somewhat counter-intuitively, the smallest GatedGCN model, with 2 layers and 16 hidden features per layer, continued to outperform the non-learnt heuristic baseline and all other GNN architectures.

Explicitly maintaining edge features. To dissect the unreasonable effectiveness of GatedGCNs for edge tasks, we change the architecture's node update equations to not explicitly maintain an edge feature $\hat{e}_{ij}$ across layers, *i.e.*, we replace Eq.(24) as:

$$\hat{e}_{ij}^\ell = \mathrm{ReLU}\Big(A^\ell h_i^{\ell-1} + B^\ell h_j^{\ell-1}\Big). \tag{38}$$

GatedGCN with Eq.(38) is similar to other anisotropic GNN variants, GAT and MoNet, which do not explicitly maintain an edge feature representation across layers.

We find that maintaining edge features is important for performance, especially for smaller models. Larger GatedGCNs without explicit edge features do not achieve the same performance as having edge features, but are still able to outperform the non-learnt heuristic. It will be interesting to further analyze the importance and trade-offs of maintaining edge features for real-world edge classification tasks.

E.2. GraphSage: Mean and Maxpool variants

The Max-pool variant of GraphSage, Eq.(12), which adds an additional linear transformation to neighborhood features and takes an element-wise maximum across them, is isotropic but should be more powerful than the Mean variant, Eq.(10). For our main experiments, we used GraphSage-Mean in order to motivate our aim of identifying the basic building blocks for *upgrading* the GCN architecture. Empirically, we found GraphSage-Max to have similar performance as GraphSage-Mean on TSP, see Table 14. In future work, it would be interesting to further study the application

Table 13. Performance on TSP test set graphs for the GatedGCN architecture at various model scales and with/without explicit edge representations. All models are augmented with residual connections and normalization. **Bold** indicates the best model between the two edge representation equations.

L	d	**With edge repr: Eq.(24)**			**Without edge repr: Eq.(38)**		
		#Param	**F1**	**Epoch/Total**	**#Param**	**F1**	**Epoch/Total**
2	16	3610	**0.744**	110.54s/10.04hr	2970	0.624	119.91s/6.66hr
4	32	24434	**0.779**	149.24s/11.11hr	19890	0.722	158.04s/8.78hr
4	64	94946	**0.790**	203.07s/16.31hr	77666	0.752	184.79s/10.27hr
32	64	684514	**0.843**	1760.56s/48.00hr	547170	0.783	1146.92s/48.00hr
k-NN Heuristic, k=2, F1: 0.693							

Table 14. Performance on TSP test set graphs for GraphSage variants with and without residual connections (higher is better). Results are averaged over 2 runs with 2 different seeds. **Bold** indicates the best model between residual and non-residual connections.

Model	**#Param**	**Residual**		**No Residual**	
		F1	**Epoch/Total**	**F1**	**Epoch/Total**
GraphSage (Mean)	98450	**0.663±0.003**	145.75s/16.05hr	0.657±0.002	147.22s/14.33hr
GraphSage (Max)	94522	0.664±0.001	155.76s/12.94hr	**0.667**±0.002	156.23s/11.75hr
k-NN Heuristic	k=2	F1: 0.693			

of linear transformations before/after neighborhood aggregation as well as the trade-offs between various aggregation functions.

F. Note on DiffPool

In our main experiments, we fix a parameter budget of 100k and arbitrarily select 4 GNN layers. Then, we estimate the model hyper-parameters to match the budget. However, for DiffPool (Ying et al., 2018), the number of trainable parameters is over 100k in one experiment—TU-DD[11]. Apparently, the minimum requirement to constitute a DiffPool model is a single differentiable pooling layer, preceded and followed by a number of GNN layers. Thus, DiffPool effectively uses more GNN layers compared to all other models, illustrated in Figure 12. Following Ying et al. (2018), we use GraphSage at each level of hierarchy and for the pooling.

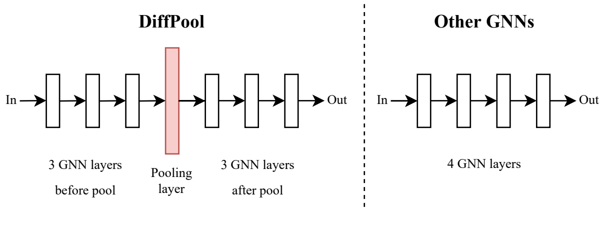


Figure 12. Model depth for DiffPool compared to other GNNs. A maximum of 4 layers are used for all GNN models. For DiffPool, 3 GNN layers are used before and after the pooling layer.

Table 15. Performance on the standard TU test sets for DiffPool. The label *-big* denotes models having hidden features increased to 32 at each layer, and *-budget* denotes the original model reported in the main paper.

Dataset	**Model**	**#Param**	**seed 1**		**seed 2**	
			Acc ± s.d.	**Epoch/Total**	**Acc ± s.d.**	**Epoch/Total**
DD	DiffPool-*big*	592230	76.54±2.90	39.41s/33.49hr	75.91±2.76	37.83s/32.65hr
	DiffPool-*budget*	165342	65.91±9.45	37.87s/33.42hr	63.73±1.49	37.48s/32.87hr
PROTEINS	DiffPool-*big*	137390	77.30±2.69	4.01s/4.09hr	76.70±4.20	3.96s/3.97hr
	DiffPool-*budget*	93780	76.60±2.11	3.99s/3.93hr	75.90±2.88	3.99s/3.90hr

In order to be as close as possible to the budget of 100k parameters for DiffPool, we set the hidden dimension to be significantly smaller than other GNNs (*e.g.*, as few as 8 for DD, compared to 128 for GCN). Despite smaller hidden dimensions, DiffPool still has the highest trainable parameters per experiment due to increased depth as well as the dense pooling layer. However, DiffPool performs poorly compared to other GNNs on the TU datasets, see Table 2. We attribute its low performance to the small hidden dimension.

For completeness sake, we increase the hidden dimensions for DiffPool to 32, which raises the parameter count to 592k for DD. Our results for DD and PROTEINS, presented in Table 15, show that larger DiffPool models match the performance of other GNNs (Table 2). Nonetheless, our goal is not to find the optimal set of hyper-parameters for a model, but to identify performance trends and important mechanisms for designing GNNs. In future work, it would be interesting to further study the design of hierarchical representation learning methods such as DiffPool.

G. Note on Model Timings and Hardware

Timing research code can be tricky due to differences of implementations and hardware acceleration, *e.g.*, our implementations of GAT can be optimized by taking a parallelized approach for *multi-head* computation. Similarly, MoNet can be improved by pre-computing the *in-degrees* of batched graph nodes that are used as *pseudo* edge features to compute gaussian weights. Somewhat counter-intuitively, our GIN implementation is significantly faster than all other models, including *vanilla* GCN. Nonetheless, we take a practical view and report the average wall clock time per epoch and the total training time for each model. All experiments were implemented in DGL/PyTorch. We run experiments for TU, MNIST, CIFAR10, ZINC and TSP on an Intel Xeon CPU E5-2690 v4 server with 4 Nvidia 1080Ti GPUs, and for PATTERN and CLUSTER on an Intel Xeon Gold 6132 CPU with 4 Nvidia 2080Ti GPUs. Each experiment was run on a single GPU and 4 experiments were run on the server at any given time (on different GPUs). We run each experiment for a maximum of 48 hours.

[11] Note that the input feature vector for each node in DD graphs is larger than other datasets, see Table 10.

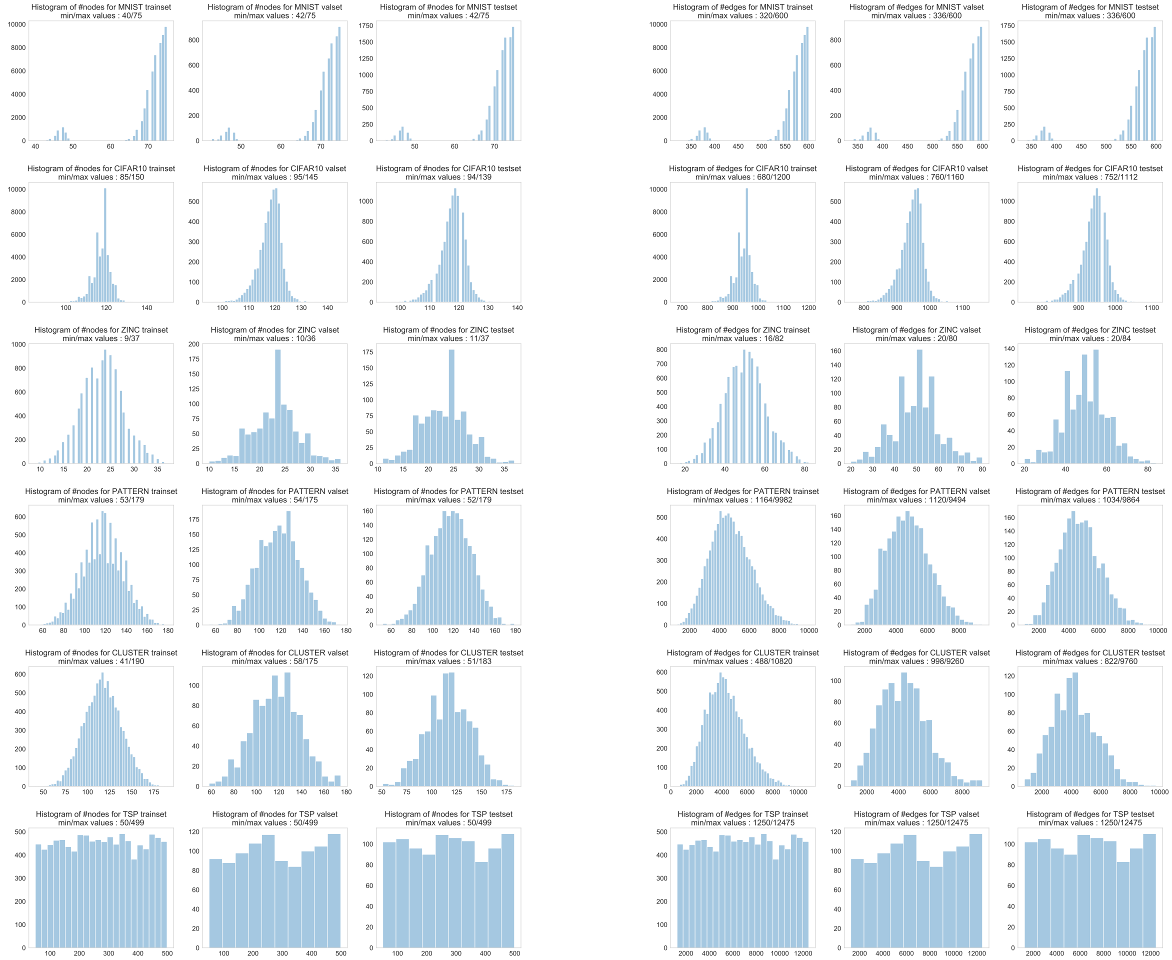


Figure 13. Histograms of nodes (left column) and edges (right column) per graph in MNIST, CIFAR10, ZINC, PATTERN, CLUSTER and TSP datasets.

H. Hyperparameter Settings

The hyperparameter settings for all models in the main paper across all benchmarked datasets are provided in Table 16.

Table 16. Hyperparameter settings for all experiments. L is the number of layers (In DiffPool, L defines the number of GNN layers in each block before or after pooling); *hidden* and *out* are the number of hidden and output features respectively; *init lr* is the initial learning rate, *patience* is the decay patience, *min lr* is the stopping lr, all experiments have *lr reduce factor* 0.5; *#Params* is the number of training parameters.

Dataset	**Model**	**Hyperparameters**				**Learning Setup**			**#Params**
		L	hidden	out	Other	init lr	patience	min lr	
ENZYMES	MLP	4	128	128	gated:False; readout:mean	1e-3	25	1e-6	62502
	MLP (Gated)	4	128	128	gated:True; readout:mean	1e-3	25	1e-6	79014
	GCN	4	128	128	readout:mean	7e-4	25	1e-6	80038
	GraphSage	4	96	96	sage_aggregator:meanpool; readout:mean	7e-4	25	1e-6	82686
	GIN	4	96	96	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum; readout:sum	7e-3	25	1e-6	80770
	DiffPool	3	64	–	embedding_dim:64; sage_aggregator:meanpool; num_pool:1; pool_ratio:0.15; linkpred:True; readout:mean	7e-3	25	1e-6	94782
	GAT	4	16	128	n_heads:8; readout:mean	1e-3	25	1e-6	80550
	MoNet	4	80	80	kernel:3; pseudo_dim_MoNet:2; readout:mean	1e-3	25	1e-6	83538
	GatedGCN	4	64	64	edge_feat:False; readout:mean	7e-4	25	1e-6	89366
DD	MLP	4	128	128	gated:False; readout:mean	1e-4	25	1e-6	71458
	MLP (Gated)	4	128	128	gated:True; readout:mean	1e-4	25	1e-6	87970
	GCN	4	128	128	readout:mean	1e-5	25	1e-6	88994
	GraphSage	4	96	96	sage_aggregator:meanpool; readout:mean	1e-5	25	1e-6	89402
	GIN	4	96	96	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum; readout:sum	1e-3	25	1e-6	85646
	DiffPool	3	8	–	embedding_dim:8; sage_aggregator:meanpool; num_pool:1; pool_ratio:0.15; linkpred:True; readout:mean	5e-4	25	1e-6	165342
	GAT	4	16	128	n_heads:8; readout:mean	5e-5	25	1e-6	89506
	MoNet	4	80	80	kernel:3; pseudo_dim_MoNet:2; readout:mean	7e-5	25	1e-6	89134
	GatedGCN	4	64	64	edge_feat:False; readout:mean	1e-5	25	1e-6	98386
PROTEINS	MLP	4	128	128	gated:False; readout:mean	1e-4	25	1e-6	63778
	MLP (Gated)	4	128	128	gated:True; readout:mean	1e-4	25	1e-6	80290
	GCN	4	128	128	readout:mean	7e-4	25	1e-6	81314
	GraphSage	4	96	96	sage_aggregator:meanpool; readout:mean	7e-5	25	1e-6	83642
	GIN	4	96	96	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum; readout:sum	7e-3	25	1e-6	79886
	DiffPool	3	22	–	embedding_dim:22; sage_aggregator:meanpool; num_pool:1; pool_ratio:0.15; linkpred:True; readout:mean	1e-3	25	1e-6	93780
	GAT	4	16	128	n_heads:8; readout:mean	1e-3	25	1e-6	81826
	MoNet	4	80	80	kernel:3; pseudo_dim_MoNet:2; readout:mean	7e-5	25	1e-6	84334
	GatedGCN	4	64	64	edge_feat:False; readout:mean	1e-4	25	1e-6	90706
MNIST	MLP	4	168	168	gated:False; readout:mean	1e-3	5	1e-5	104044
	MLP (Gated)	4	150	150	gated:True; readout:mean	1e-3	5	1e-5	105717
	GCN	4	146	146	readout:mean	1e-3	5	1e-5	101365
	GraphSage	4	108	108	sage_aggregator:meanpool; readout:mean	1e-3	5	1e-5	102691
	GIN	4	110	110	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum; readout:sum	1e-3	5	1e-5	105434
	DiffPool	3	32	–	embedding_dim:32; sage_aggregator:meanpool; num_pool:1; pool_ratio:0.15; linkpred:True; readout:mean	1e-3	5	1e-5	106538
	GAT	4	19	152	n_heads:8; readout:mean	1e-3	5	1e-5	110400
	MoNet	4	90	90	kernel:3; pseudo_dim_MoNet:2; readout:mean	1e-3	5	1e-5	104049
	GatedGCN	4	70	70	edge_feat:False (edge_feat:True for GatedGCN-E); readout:mean	1e-3	5	1e-5	104217
CIFAR10	MLP	4	168	168	gated:False; readout:mean	1e-3	5	1e-5	104044
	MLP (Gated)	4	150	150	gated:True; readout:mean	1e-3	5	1e-5	106017
	GCN	4	146	146	readout:mean	1e-3	5	1e-5	101657
	GraphSage	4	108	108	sage_aggregator:meanpool; readout:mean	1e-3	5	1e-5	102907
	GIN	4	110	110	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum; readout:sum	1e-3	5	1e-5	105654
	DiffPool	3	32	–	embedding_dim:16; sage_aggregator:meanpool; num_pool:1; pool_ratio:0.15; linkpred:True; readout:mean	1e-3	5	1e-5	108042
	GAT	4	19	152	n_heads:8; readout:mean	1e-3	5	1e-5	110704
	MoNet	4	90	90	kernel:3; pseudo_dim_MoNet:2; readout:mean	1e-3	5	1e-5	104229
	GatedGCN	4	70	70	edge_feat:False (edge_feat:True for GatedGCN-E); readout:mean	1e-3	5	1e-5	104357
ZINC	MLP	4	150	150	gated:False; readout:mean	1e-3	5	1e-5	108975
	MLP (Gated)	4	135	135	gated:True; readout:mean	1e-3	5	1e-5	106970
	GCN	4	145	145	readout:mean	1e-3	5	1e-5	103077
	GraphSage	4	108	108	sage_aggregator:meanpool; readout:mean	1e-3	5	1e-5	105031
	GIN	4	110	110	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum; readout:sum	1e-3	5	1e-5	103079
	DiffPool	3	56	–	embedding_dim:56; sage_aggregator:meanpool; num_pool:1; pool_ratio:0.15; linkpred:True; readout:mean	1e-3	5	1e-5	110561
	GAT	4	18	144	n_heads:8; readout:mean	1e-3	5	1e-5	102385
	MoNet	4	90	90	kernel:3; pseudo_dim_MoNet:2; readout:mean	1e-3	5	1e-5	106002
	GatedGCN	4	70	70	edge_feat:False (edge_feat:True for GatedGCN-E); readout:mean	1e-3	5	1e-5	105735
PATTERN	MLP	4	150	150	gated:False; readout:mean	1e-3	5	1e-5	105263
	MLP (Gated)	4	135	135	gated:True; readout:mean	1e-3	5	1e-5	103629
	GCN	4	146	146	readout:mean	1e-3	5	1e-5	100923
	GraphSage	4	106	106	sage_aggregator:meanpool; readout:mean	1e-3	5	1e-5	98607
	GIN	4	110	110	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum; readout:sum	1e-3	5	1e-5	100884
	GAT	4	19	152	n_heads:8; readout:mean	1e-3	5	1e-5	109936
	MoNet	4	90	90	kernel:3; pseudo_dim_MoNet:2; readout:mean	1e-3	5	1e-5	103775
	GatedGCN	4	70	70	edge_feat:False; readout:mean	1e-3	5	1e-5	104003
CLUSTER	MLP	4	150	150	gated:False; readout:mean	1e-3	5	1e-5	106015
	MLP (Gated)	4	135	135	gated:True; readout:mean	1e-3	5	1e-5	104305
	GCN	4	146	146	readout:mean	1e-3	5	1e-5	101655
	GraphSage	4	106	106	sage_aggregator:meanpool; readout:mean	1e-3	5	1e-5	99139
	GIN	4	110	110	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum; readout:sum	1e-3	5	1e-5	103544
	GAT	4	19	152	n_heads:8; readout:mean	1e-3	5	1e-5	110700
	MoNet	4	90	90	kernel:3; pseudo_dim_MoNet:2; readout:mean	1e-3	5	1e-5	104227
	GatedGCN	4	70	70	edge_feat:False; readout:mean	1e-3	5	1e-5	104355
TSP	MLP	3	144	144	gated:False; readout:mean	1e-3	10	1e-5	94394
	MLP (Gated)	3	144	144	gated:True; readout:mean	1e-3	10	1e-5	115274
	GCN	4	128	128	readout:mean	1e-3	10	1e-5	108738
	GraphSage	4	96	96	sage_aggregator:meanpool; readout:mean	1e-3	10	1e-5	98450
	GIN	4	80	80	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum; readout:sum	1e-3	10	1e-5	118574
	GAT	4	16	128	n_heads:8; readout:mean	1e-3	10	1e-5	109250
	MoNet	4	80	80	kernel:3; pseudo_dim_MoNet:2; readout:mean	1e-3	10	1e-5	94274
	GatedGCN	4	64	64	edge_feat:False (edge_feat:True for GatedGCN-E); readout:mean	1e-3	10	1e-5	94946