Sparsifying the Update Step in Graph Neural Networks

Johannes Lutzeyer, Changmin Wu, Michalis Vazirgiannis

Sparsifying the Update Step in Graph Neural Networks

Johannes F. Lutzeyer*
Ecole Polytechnique, Palaiseau, France
johannes.lutzeyer@polytechnique.edu

Changmin Wu*
Ecole Polytechnique, Palaiseau, France
changmin.wu@polytechnique.edu

Michalis Vazirgiannis
Ecole Polytechnique, Palaiseau, France
AUEB, Greece
mvazirg@lix.polytechnique.fr

Abstract

Message-Passing Neural Networks (MPNNs), the most prominent Graph Neural Network (GNN) framework, celebrate much success in the analysis of graph-structured data. Concurrently, the sparsification of Neural Network models attracts a great amount of academic and industrial interest. In this paper we conduct a structured study of the effect of sparsification on the trainable part of MPNNs known as the Update step. To this end, we design a series of models to successively sparsify the linear transform in the Update step. Specifically, we propose the ExpanderGNN model with a tuneable sparsification rate and the Activation-Only GNN, which has no linear transform in the Update step. In agreement with a growing trend in the literature the sparsification paradigm is changed by initialising sparse neural network architectures rather than expensively sparsifying already trained architectures. Our novel benchmark models enable a better understanding of the influence of the Update step on model performance and outperform existing simplified benchmark models such as the Simple Graph Convolution. The ExpanderGNNs, and in some cases the Activation-Only models, achieve performance on par with their vanilla counterparts on several downstream tasks, while containing significantly fewer trainable parameters. In experiments with matching parameter numbers our benchmark models outperform the state-of-the-art GNN models. Our code is publicly available at: https://github.com/ChangminWu/ExpanderGNN.

1 Introduction

In recent years we have witnessed the blossom of Graph Neural Networks (GNNs). They have become the standard tools for analysing and learning graph-structured data (Wu et al. 2020) and have demonstrated convincing performance in various application areas, including chemistry (Duvenaud et al. 2015), social networks (Monti et al. 2019), natural language processing (Yao et al. 2019) and neural science (Griffa et al. 2017).

Among various GNN models, Message-Passing Neural Networks (MPNNs, Gilmer et al. (2017)) and their variants are considered to be the dominating class. In MPNNs, the learning procedure can be separated into three major steps: *Aggregation*, *Update* and *Readout*, where *Aggregation* and *Update* are repeated iteratively so that each node's representation is updated recursively based on the transformed information aggregated over its neighbourhood. There is thus a division of labour

*Both authors contributed equally to this research.

Preprint. Under review.

between the *Aggregation* and the *Update* step, where the *Aggregation* utilises local graph structure, while the *Update* step is only applied to single node representations at a time independent of the local graph structure. From this a natural question then arises: *What is the impact of the graph-agnostic Update step on the performance of GNNs?* Since the *Update* step is the main source of model parameters in MPNNs, understanding its impact is fundamental in the design of parsimonious GNNs.

Wu et al. (2019) first challenged the role of the *Update* step by proposing a Simple Graph Convolution (SGC) model where they removed the non-linearities in the *Update* steps and collapsed the consecutive linear transforms into a single transform. Their experiments showed, surprisingly, that in some instances the *Update* step of Graph Convolutional Network (GCN, Kipf & Welling (2017)) can be left out completely without the models' accuracy decreasing.

In the same spirit, we propose in this paper to analyse the impact of the *Update* step and its sparsification in a systematic way. To this end, we propose two nested model classes, where the *Update* step is successively sparsified. In the first model class which we refer to as *Expander*GNN, the linear transform layers of the *Update* step are sparsified; while in the second model class, the linear transform layers are removed and only the activation functions remain in the model. We name the second model *Activation-Only* GNN and it contrasts the SGC where the activation functions where removed to merge the linear layers.

Inspired by the recent advances in the literature of sparse Convolutional Neural Network (CNN) architectures (Prabhu et al. 2018), we propose to utilise a random sparsification scheme, which is motivated by the study of expander graphs (hence the model's name). Here the sparsification is performed at intialisation and accordingly saves the cost of more traditional methods, which often iteratively prune connections during training.

Through a series of empirical assessments on different graph learning tasks (graph and node classification as well as graph regression), we demonstrate that the *Update* step can be heavily simplified without inhibiting performance or relevant model expressivity. Our findings partly agree with the work in Wu et al. (2019), in that dense *Update* steps in GNN are expensive and often ineffectual. In contrast to their proposition, we find that there are many instances in which leaving the *Update* step out completely significantly harms performance. In these instances our *Activation-Only* model shows superior performance while matching the number of parameters and efficiency of the SGC.

Our contributions can be summarised as follows.

1. We explore the impact of the *Update* step and its sparsification in MPNNs through the newly proposed model class of *Expander*GNNs with tuneable density. We show empirically that *a sparse Update step matches the performance of the standard model architectures*.
2. As an extreme case of the *Expander*GNN, as well as an alternative to the SGC, we propose the *Activation-Only* GNNs that remove the linear transformation layer from the *Update* step and keep non-linearity in tact. We observe the *Activation-Only* models to exhibit comparable, sometimes significantly superior performance to the SGC while being equally time and memory efficient.

Both of our proposed model classes can be extrapolated without further efforts to a variety of models in the MPNN framework and hence provide practitioners with an array of efficient and often highly performant benchmark models.

The rest of this paper is organised as follows. In Section 2, we provide an overview of the related work. Section 3 introduces preliminary concepts of MPNNs, followed by a detailed presentation of our two proposed model classes. Section 4 discusses our experimental setting and empirical evaluation of the proposed models in a variety of downstream graph learning tasks.

2 Related Work

In recent years the idea of *utilising expander graphs in the design of neural networks* is starting to be explored in the CNN literature. Most notably, Prabhu et al. (2018) propose to replace linear fully connected layers in deep networks using an expander graph sampling mechanism and hence, propose a novel CNN architecture they call X-nets. The great innovation of this approach is that well-performing sparse neural network architectures are initialised rather than expensively calculated. Furthermore, they are shown to compare favourably in training speed, accuracy and performance

trade-offs to several other state-of-the-art architectures. McDonald & Shokoufandeh (2019) and Kepner & Robinett (2019) build on the X-net design and propose alternative expander sampling mechanisms to extend the simplistic design chosen in the X-nets. Independent of this literature branch, Bourely et al. (2017) explore 6 different mechanisms to randomly sample expander graph layers. Across the literature the results based on expander graph layers are encouraging.

The *Sparisification and Pruning of neural networks* is a very active research topic (Hoefler et al. 2021, Blalock et al. 2020). In particular, a wealth of algorithms sparsifying neural network architectures at initialisation has recently been proposed (Tanaka et al. 2020, Wang, Zhang & Grosse 2020, Lee et al. 2019). While these algorithms make pruning decisions on a per-weight basis, Frankle et al. (2021) find that these algorithms produce equivalent results to a per-layer choice of a fraction of weights to prune, as is directly done in our chosen sparsification scheme. All of these research efforts are pruning CNNs, typically the VGG and ResNet architectures. *To the best of our knowledge, our work is the first investigating the potential of sparsifying the trainable parameters in GNNs.*

Both Wu et al. (2019) and Salha et al. (2019) observed that *simplifications in the Update step of the GCN model* is a promising area of research. Wu et al. (2019) proposed the SGC model, where simplification is achieved by removing the non-linear activation functions from the GCN model. This removal allows them to merge all linear transformations in the *Update* steps into a single linear transformation without sacrificing expressive power. Salha et al. (2019) followed a similar rationale in their simplification of the graph autoencoder and variational graph autoencoder models. These works have had an immediate impact on the literature featuring as benchmark models and object of study in many recent papers: The idea of omitting the *Update* step guided Chen et al. (2020) in the design of simplified models and has found successful application in various areas where model complexity needs to be reduced (Waradpande et al. 2020, He et al. 2020) or very large graphs need to be processed (Salha et al. 2020). In our work we aim to extend these efforts by providing more simplified benchmark models for GNNs without a specific focus on the GCN.

3 Investigating the Role of the Update step

In this section, we present the two proposed model classes, where we sparsify or remove the linear transform layer in the *Update* step, with the aim to systematically analyse the impact of the *Update* step. We begin in Section 3.1 by introducing the general model structure of MPNNs, the GNN class we study in this paper. We then demonstrate how the *Expander*GNN and *Activation-Only* GNN are constructed in Sections 3.2 and 3.3, respectively.

3.1 Preliminaries

Message-Passing Graph Neural Networks We define graphs $\mathcal{G} = (\boldsymbol{A}, \boldsymbol{X})$ in terms of their adjacency matrix $\boldsymbol{A} = [0,1]^{n \times n}$, which contains the information of the graph's node set $\mathbb{V}$, and the node features $\boldsymbol{X} \in \mathbb{R}^{n \times s}$. Given a graph $\mathcal{G}$, a graph learning task aims at learning meaningful embeddings on the node or graph level that can be used in downstream tasks such as node or graph classification. MPNNs, a prominent paradigm that arose in recent years for performing machine learning tasks on graphs, learn such embeddings by iteratively aggregating information from the neighbourhoods of each node and updating their representations based on this information. Precisely, the learning procedure of MPNNs can be divided into the following phases:

Initial (optional). In this phase, the initial node features $\boldsymbol{X}$ are mapped from the feature space to a hidden space by a parameterised neural network $U^{(0)}$, usually a fully-connected linear layer.

$$\boldsymbol{H}^{(1)} = U^{(0)}(\boldsymbol{X}) = \left(\boldsymbol{h}_1^{(1)}, \ldots, \boldsymbol{h}_n^{(1)}\right),$$

where the hidden representation of node i is denoted as $\boldsymbol{h}_i^{(1)}$, which will be used as the initial point for later iterations.

Aggregation. In this phase, MPNNs gather, for each node, information from the node's neighbourhood, denoted $\mathcal{N}(i)$ for node i. The gathered pieces of information are called "messages", denoted by $\boldsymbol{m}_i$. Formally, if $f^{(l)}(\cdot)$ denotes the aggregation function at iteration l, then

$$\boldsymbol{m}_i^{(l)} = f^{(l)}\left(\left\{\boldsymbol{h}_j^{(l)} | j \in \mathcal{N}(i)\right\}\right).$$

Due to the isotropic nature of graphs (arbitrary node labelling), this function needs to be permutation equivariant or invariant. It also has to be differentiable so that the framework will be end-to-end trainable.

Update. The nodes then update their hidden representations based on their current representations and the received "messages". Let $U^{(l)}$ denote the update function at iteration l. For node i, we have

$$\boldsymbol{h}_i^{(l+1)} = U^{(l)}\left(\boldsymbol{h}_i^{(l)}, \boldsymbol{m}_i^{(l)}\right).$$

Readout (optional). After L aggregation and update iterations, depending on the downstream tasks, the MPNN will either output node representations directly or generate a graph representation via a differentiable readout function,

$$\boldsymbol{g} = R\left(\left\{\boldsymbol{h}_i^{(L)} | i \in \mathbb{V}\right\}\right).$$

Note that various choices of *Aggregation*, *Update* and *Readout* functions are proposed in the literature. To avoid shifting from the subject of this paper, we work with the simplest and most widely used function choices, such as sum, mean and max aggregators for *Aggregation*, the Multi-Layer Perceptron (MLP) for the *Update* step and summation for the *Readout*. As an example to visualise our models in Sections 3.2 and 3.3 we use the following matrix representation of the GCN's model equation,

$$\boldsymbol{H}^{(L)} = \sigma\left(\hat{\boldsymbol{A}} \ldots \sigma\left(\hat{\boldsymbol{A}}\boldsymbol{H}^{(1)}\boldsymbol{W}^{(1)}\right) \ldots \boldsymbol{W}^{(L)}\right), \tag{1}$$

where σ denotes a nonlinear activation function, $\boldsymbol{W}^{(i)}$ contains the trainable weights of the linear transform in the *Update* step and $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ is the symmetric normalised adjacency matrix with $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ denoting the adjacency matrix with added self-loops and $\tilde{\boldsymbol{D}}$ being the corresponding degree matrix.

3.2 Sparsifying the Update step: Expander GNN

In this section we propose the *Expander*GNN model where sampled expander graphs are used to initialise sparse linear layers in the *Update* step. We begin by discussing how linear layers in a neural networks can be represented by bipartite graphs.

Linear Layer as a graph The fully-connected linear transform layer in MLPs can be represented by a bipartite graph $\mathcal{B}(\mathbb{S}_1, \mathbb{S}_2, \mathbb{E})$, where $\mathbb{S}_1$ and $\mathbb{S}_2$ are two sets of nodes and $\mathbb{E}$ the set of edges that satisfy $\forall u \in \mathbb{S}_1, \forall v \in \mathbb{S}_2, \exists (u, v) \in \mathbb{E}$; $\quad \forall u, v \in \mathbb{S}_1 (\text{resp.} \mathbb{S}_2), \nexists (u, v) \in \mathbb{E}$. The number of edges, i.e., parameters, is $|\mathbb{S}_1||\mathbb{S}_2|$ and the edges can be encoded in matrix form by $\boldsymbol{W} \in \mathbb{R}^{|\mathbb{S}_1| \times |\mathbb{S}_2|}$, the weight matrix in (1), that maps the input node features of dimension $|\mathbb{S}_1|$ to output node features of dimension $|\mathbb{S}_2|$.

Expander Linear Layer Given the bipartite graph corresponding to a linear transform layer $\mathcal{B}(\mathbb{S}_1, \mathbb{S}_2, \mathbb{E})$, we follow the design of Prabhu et al. (2018) to construct the sparsifier by sampling its subgraph of specific expander structure.

Definition 1. *Suppose $|\mathbb{S}_1| \leq |\mathbb{S}_2|$. For each vertex $u \in \mathbb{S}_1$, we uniformly sample d vertices $\{v_i^u\}_{i=1,\ldots,d}$ from $\mathbb{S}_2$ to be connected to u. Then, the constructed graph $\mathcal{B}'(\mathbb{S}_1, \mathbb{S}_2, \mathbb{E}')$ is a subgraph of $\mathcal{B}$ with edge set $\mathbb{E}' = \{(u, v_i^u) : u \in \mathbb{S}_1, i \in \{1, \ldots, d\}\}$. Else if $|\mathbb{S}_1| > |\mathbb{S}_2|$, we define the expander sparsifier with the roles of $\mathbb{S}_1$ and $\mathbb{S}_2$ reversed meaning that we sample nodes from $\mathbb{S}_1$. We call a linear layer with a computational graph $\mathcal{B}'(\mathbb{S}_1, \mathbb{S}_2, \mathbb{E}')$ an expander linear layer.*

The theoretical computational cost of an expander linear layer is equal to $2nd\min(|\mathbb{S}_1|, |\mathbb{S}_2|)$ Floating Point Operations (FLOPs). The tunable parameter d can therefore lead to significant computational savings as the computational cost of a fully connected linear layer equals $2n|\mathbb{S}_1||\mathbb{S}_2|$ FLOPs.

We refer to the *density* of the expander linear layer as the ratio of the number of sampled connections to the number of connections in the complete bipartite graph. For example, the fully-connected layer has density 1. The sampling scheme in Definition 1 returns an expander linear layer of density $d/\max(|\mathbb{S}_1|, |\mathbb{S}_2|)$.

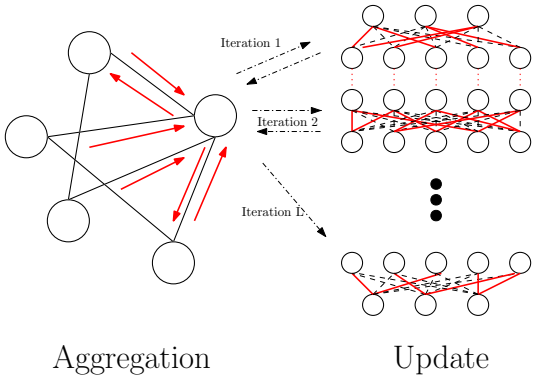

Figure 1: Illustration of the main computational steps in *Expander*GNNs. **(Left)** *Aggregation* or graph propagation step and **(Right)** *Update* step. The red lines in the *Update* step represent preserved connections in MLPs sampled as expander sparsifier structures. In the *Aggregation* step only a subset of the exchanged messages are illustrated.

When we replace all linear layers in the *Update* steps of a GNN with expander linear layers constructed by the sampling scheme in Definition 1, we get the *Expander*GNN. An illustration can be found in Figure 1.

When compared to pruning algorithms which sparsify neural network layers by iteratively removing parameters according to certain metric during training, the expander sparsifiers have two advantages:

1. The expander design assures that paths exist between consecutive layers, avoiding the risk of *layer-collapse* that is common in many pruning algorithms, where the algorithm prunes all parameters (weights) in one layer and cuts down the flow between input and output (Tanaka et al. 2020).
2. The expander sparsifier removes parameters at initialisation and keeps the sparsified structures fixed during training, which avoids the expensive computational cost stemming from adapting the neural network architecture during or after training and then retraining the network as is done in the majority of pruning algorithms (Frankle & Carbin 2019, Han et al. 2015).

Motivation of Expander Linear Layer The sampling scheme in Definition 1 samples bipartite graphs with good expansion properties, which are commonly discussed in the field of error correcting codes under the name "lossless expanders" (Hoory et al. 2006, pp. 517-522). Expander graphs can be informally defined to be highly connected and sparse graphs (Lubotzky 2012). They are successfully applied in communication networks where communication comes at a certain cost and is to be used such that messages are spread across the network efficiently (Lubotzky 2012). Equally, in a neural network each parameter (corresponding to an edge in the neural network architecture) incurs a computational cost and is placed to optimise the overall performance of the neural network architecture. Therefore, the use of expander graphs in the design of neural network architectures is conceptually well motivated.

In Bölcskei et al. (2019) the connectedness of a sparse neural network architecture was linked to the complexity of a given function class which can be approximated by sparse neural networks. Hence, utilising neural network parameters to optimise the connectedness of the network maximises the expressivity of the neural network. In Kepner & Robinett (2019) and Bourely et al. (2017) the connectedness of the neural network architecture graph was linked – via the path-connectedness and the graph Laplacian eigenvalues – to the performance of neural network architectures. Therefore, for both the expressivity of the neural network and its performance, the connectedness, which is optimised in expander graphs, is a parameter of interest.

Implementation of Expander Linear Layer The most straightforward way of implementing the expander linear layer is to store the weight matrix $\boldsymbol{W}$ as a sparse matrix. Sparse matrix multiplications can be accelerated on several processing units released in 2020 and 2021 such as the Sparse Linear Algebra Compute (SLAC) cores used in the Cerebras WSE-2 (CEREBRAS SYSTEMS 2021), the Intelligence Processing Unit (IPU) produced by Graphcore (Strategy 2020) and the NVIDIA A100

Table 1: Model Equations of the Vanilla, *Expander* and *Activation-Only* GNNs.

Model		Aggregation	Update	Remarks
GCN	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = \frac{1}{\sqrt{d_i}} \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)} \frac{1}{\sqrt{d_j}}$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$	1. **GCN:** d_i denotes the degree of node i.
	Activation-Only		$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{m}_i^{(l)}\right)$	2. **GIN:** ϵ is a learnable ratio added explicitly to the central node's own representation.
GIN	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = (1+\epsilon)\boldsymbol{h}_i^{(l)} + \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$	3. **PNA:** $\bigoplus$ corresponds to an operator formed by taking the tensor product of a vector containing three scalar functions and four aggregator functions, resulting in a tensor indexed by i, s, a, where the index i corresponds to the currently considered node, s corresponds to the scalar dimension and a indexes the aggregator dimension. For more details see Corso et al. (2020).
	Activation-Only		$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{m}_i^{(l)}\right)$	
GraphSage	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = \text{CONCAT}\left(\boldsymbol{h}_i^{(l)}, \text{MAX}_{j \in \mathcal{N}(i)} \sigma(\boldsymbol{h}_j^{(l)} \boldsymbol{M}_1^{(l)} \odot \boldsymbol{W}_1^{(l)})\right)$	$\boldsymbol{h}_i^{(l+1)} = \frac{\sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}_2^{(l)} \odot \boldsymbol{W}_2^{(l)})}{\|\sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}_2^{(l)} \odot \boldsymbol{W}_2^{(l)})\|_2}$	
	Activation-Only	$\boldsymbol{m}_i^{(l)} = \boldsymbol{h}_i^{(l)} + \text{MAX}_{j \in \mathcal{N}(i)} \sigma(\boldsymbol{h}_j^{(l)})$	$\boldsymbol{h}_i^{(l+1)} = \frac{\sigma(\boldsymbol{m}_i^{(l)})}{\|\sigma(\boldsymbol{m}_i^{(l)})\|_2}$	
PNA	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = \text{CONCAT}_{s,a}\left(\bigoplus_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}\right)$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$	
	Activation-Only	$\boldsymbol{m}_i^{(l)} = \frac{1}{12} \sum_{s,a} \left[\bigoplus_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}\right]_{i,s,a}$	$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{m}_i^{(l)}\right)$	
MLP	Vanilla/*Expander*	$\boldsymbol{m}^{(l)} = \boldsymbol{h}_i^{(l)}$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$	

Tensor Core GPU (NVIDIA 2020). However, since we ran experiments on a NVIDIA RTX 2060 GPU, we use masks in our implementation, similar to those of several existing pruning algorithms, to achieve the sparsification. A mask $\boldsymbol{M} \in \{0,1\}^{|\mathbb{S}_1| \times |\mathbb{S}_2|}$ is of the same dimension as weight matrix and $M_{u,v} = 1$ if and only if $(u,v) \in \mathbb{E}'$. An entrywise multiplication, denoted by $\odot$, is then applied to the mask and the weight matrix so that undesired parameters in the weight matrix are removed, i.e., (1) can be rewritten as,

$$\boldsymbol{H}^{(L)} = \sigma\left(\hat{\boldsymbol{A}} \ldots \sigma\left(\hat{\boldsymbol{A}} \boldsymbol{H}^{(1)} \boldsymbol{M}^{(1)} \odot \boldsymbol{W}^{(1)}\right) \ldots \boldsymbol{M}^{(L)} \odot \boldsymbol{W}^{(L)}\right). \tag{2}$$

3.3 An Extreme Case: Activation-Only GNN

In Gama et al. (2020) it is argued that the non-linearity present in GNNs, in form of the activation functions, has the effect of frequency mixing in the sense that "part of the energy associated with large eigenvalues" is brought "towards low eigenvalues where it can be discriminated by stable graph filters." The theoretical insight that activation functions help capture information stored in the high energy part of graph signals is strong motivation to consider an alternative simplification to the one made in the SGC. In this alternative simplification, which we refer to as the *Activation-Only* GNN models, we remove linear transformations instead of activation functions such that each message-passing step is immediately followed by a pointwise activation function. The resulting model can be seen as a natural extension of the *Expander*GNN, where the linear transformation of the *Update* step is completely forgone. Hence, in a *Activation-Only* GNN, (1) will be rewritten as,

$$\boldsymbol{H}^{(L)} = \sigma\left(\hat{\boldsymbol{A}} \ldots \sigma\left(\hat{\boldsymbol{A}} \boldsymbol{H}^{(1)}\right)\right). \tag{3}$$

This proposed simplification is applicable to a wide variety of GNN models as we will demonstrate in our extensive set of experiments in Section 4. For comparison we display the model equation of the SGC (Wu et al. 2019),

$$\boldsymbol{H}^{(L)} = \hat{\boldsymbol{A}}^L \boldsymbol{H}^{(1)} \Theta,$$

where $\Theta = \boldsymbol{W}^{(1)} \ldots \boldsymbol{W}^{(L)}$. Here the nonlinear activation functions have been removed and the linear transformations have been collapsed into a single linear transformation layer. Interestingly, we observe that the repeated application of the symmetric matrix $\hat{\boldsymbol{A}}$ to the input data $\boldsymbol{X}$ is equivalent to an unnormalised version of the power method approximating the eigenvector corresponding to the largest eigenvalue of $\hat{\boldsymbol{A}}$. Hence, if sufficiently many layers L are used then inference is drawn in the SGC model simply on the basis of the first eigenvector of $\hat{\boldsymbol{A}}$.

4 Experiments and Discussion

In order to study the influence of the *Update* step in GNNs, we proposed a series of models in Section 3, where its linear transform is gradually sparsified. By observing the trend of model performance change (on downstream tasks) with respect to the sparsity of the linear transform layer, we measure

Table 2: Properties of all datasets used in experiments.

	Dataset	#Graphs	#Nodes (avg.)	#Edges (avg.)	Task
TU datasets	ENZYMES	600	32.63	62.14	Graph Classification
	DD	1178	284.32	715.66	
	PROTEINS	1113	39.06	72.82	
	IMDB-BINARY	1000	19.77	193.06	
Computer Vision	MNIST	70000	70.57	282.27	
	CIFAR10	60000	117.63	470.53	
	ZINC	12000	23.16	24.92	Graph Regression
Citations	CORA	1	2708	5278	Node Classification
	CITESEER	1	3327	4552	
	PUBMED	1	19717	44324	
	ogbn-arxiv	1	169343	1166243	

the impact of the *Update* step. We have made our experimentation code publicly available online[1]. In Section 4.1 we provide an overview of our experimentation setup. Then, in Sections 4.2, 4.3 and 4.4, we observe the performance of the proposed benchmark models on the tasks of graph classification, graph regression and node classification, respectively. The full set of results can be found in Appendix A. In Section 4.5, we compare the performance of *Expander*GNNs and vanilla GNNs when they have equally many parameters and in Section 4.6 we compare the convergence behaviour of the studied models.

4.1 General Settings and Baselines

Considered GNNs Throughout this section we refer to the standard, already published, architectures as "vanilla" architectures. We compare the performance of the vanilla GNN models, the *Expander*GNN models with different densities ($10\%, 50\%, 90\%$), the *Activation-Only* GNN models with different activation functions (ReLU, PReLU, Tanh), as well as the SGC for the GCN models. To ensure that our inference is not specific to a certain GNN architecture only, we evaluate the performance across 4 representative GNN models of the literature state-of-the-art. The considered models are the Graph Convolutional Network (GCN, Kipf & Welling (2017)), the Graph Isomorphism Network (GIN, Xu et al. (2019)), the GraphSage Network Hamilton et al. (2017), and the Principle Neighborhood Aggregation (PNA, Corso et al. (2020)), along with a MLP baseline that only takes the node features into account while ignoring the graph structure. The precise model equations of our proposed architectures applied to these GNNs can be found in Table 1. The *Activation-Only* model class is defined in the context of a GNN architecture and cannot be sensibly extrapolated to the MLP. Therefore, we consider only the vanilla and *Expander* variants for the MLP benchmark.

Datasets We experiment on eleven datasets from areas such as chemistry, social networks, computer vision and academic citation, for three major graph learning tasks. For graph classification, we have four TU datasets Kersting et al. (2016) which are either chemical or social network graphs, and two Image datasets (MNIST/CIFAR10) that are constructed from original images following the procedure in Knyazev et al. (2019). To perform this conversion they first extract small regions of homogeneous intensity from the images, named "Superpixels" Dwivedi et al. (2020*a*), and construct a K-nearest neighbour graph from these superpixels. The technique we implemented to extract superpixels, the choice of K and distance kernel for constructing a nearest neighbour graph are the same as in Knyazev et al. (2019) and Dwivedi et al. (2020*a*). For graph regression, we consider molecule graphs from the ZINC dataset Irwin et al. (2012). And for node classification, we use four citation datasets Sen et al. (2008), Wang, Shen, Huang, Wu, Dong & Kanakia (2020), Hu et al. (2020*a*), where the nodes are academic articles linked by citations. Details of the used datasets can be found in Table 2, where in the number of nodes and edges column we display average values if the dataset contains multiple graphs.

Experimentation Details Since we aim to observe the performance of our benchmark models independent of the GNN choice we use the model hyperparameters found to yield a fair comparison of GNN models in Dwivedi et al. (2020*a*). Specifically, we follow the same training procedure,

[1] https://github.com/ChangminWu/ExpanderGNN

such as train/valid/test dataset splits, choice of optimiser, learning rate decay scheme, as well as the same hyper-parameters, such as initial learning rate, hidden feature dimensions and number of GNN layers. We also implement the same normalisation tricks such as adding batch normalisation after non-linearity of each *Update* step. Their setting files (training procedure/hyperparameters) are made public and can be found in Dwivedi et al. (2020*b*). For the node classification task on citation datasets, we follow the settings from Wu et al. (2019). Our experiments found that the node classification task on citation graphs of small to medium size can be easily overfit and model performances heavily depend on the choice of hyperparameters. Using the same parameters with Wu et al. (2019), such as learning rate, number of training epochs and number of GNN layers, helps us achieve similar results with the paper on the same model, which allows a fair comparison between the proposed *Activation-Only* models and the SGC.

Loss functions After L message-passing iterations, we obtain

$$\boldsymbol{H}^{(L)} = \left[\boldsymbol{h}_1^{(L)}, \ldots, \boldsymbol{h}_n^{(L)}\right]^{\mathrm{T}} \in \mathbb{R}^{n \times p},$$

as the final node embedding, where we denote p as its feature dimension. Depending on the downstream task, we either keep working with $\boldsymbol{H}^{(L)}$ or construct a graph-level representation $\boldsymbol{g}$ from $\boldsymbol{H}^{(L)}$,

$$\boldsymbol{g} = \frac{1}{n} \sum_{i \in \mathbb{V}} \boldsymbol{h}_i^{(L)},$$

which we referred to as the *Readout* step in Section 3. $\boldsymbol{g}$ or $\boldsymbol{H}^{(L)}$ is then fed into a fully-connected network (MLP) to be transformed into the desired form of output for further assessment, e.g., a scalar value as a prediction score in graph regression. We denote this network as $f(\cdot)$, which, in our experiments, is fixed to be a three-layer MLP of the form

$$f(x) = \sigma(\sigma(x\boldsymbol{W}_1)\boldsymbol{W}_2)\boldsymbol{W}_3,$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{p \times (p/2)}$, $\boldsymbol{W}_2 \in \mathbb{R}^{(p/2) \times (p/4)}$, $\boldsymbol{W}_3 \in \mathbb{R}^{(p/4) \times k}$ with k being the desired output dimension. The final output, either $f(\boldsymbol{g})$ or $f(\boldsymbol{H}^{(L)})$, is compared to the ground-truth by a task-specific loss function. For graph classification and node classification, we choose cross-entropy loss and for graph regression, we use mean absolute error.

4.2 Graph Classification

Figure 2(a), (b), (c) and (d) show the experiment results of the GCN, GIN and MLP models and their *Expander* and *Activation-Only* variants on the ENZYMES, DD, PROTEINS and IMDB-BINARY datasets for graph classification. The evaluation metric is classification accuracy, where the average accuracy, obtained from a 10-folder cross validation, is used.

One direct observation from Figure 2(a), (b), (c) and (d) is that the *Expander*GNN models, even at 10% density, perform on par with the vanilla models. Surprisingly, the same is true for the *Activation-Only* model on the ENZYMES, DD and PROTEINS datasets. IMDB-BINARY is our only graph classification dataset where the node attributes are initialised to all be equal. This uninformative initialisation seems to lead to an increased performance if the linear *Update* step is present, visible in the performance gap of the *Activation-Only* models and the *Expander*GCN models. The SGC performs either on par or worse than the *Activation-Only* model.

It is known that the simple MLP can achieve better performance than GNNs (Luzhnica et al. 2019) on the several of the TU datasets. This effect is visible in Figure 2(a) and (b). Dwivedi et al. (2020*a*) show that more complex GNN models can outperform the MLP on these datasets. This known shortcoming has no impact on our conclusion, where we compare model performance within a GNN model class rather than between GNN model classes.

Figure 2(e) and (f) show the graph classification results for the MNIST and CIFAR10 datasets. The GCN *Activation-Only* model outperforms the SGC by a larger margin than we observed on the TU datasets. It seems that especially for these computer vision datasets the presence of activation functions in the GCN architecture has a large positive impact on model performance in the graph classification task.

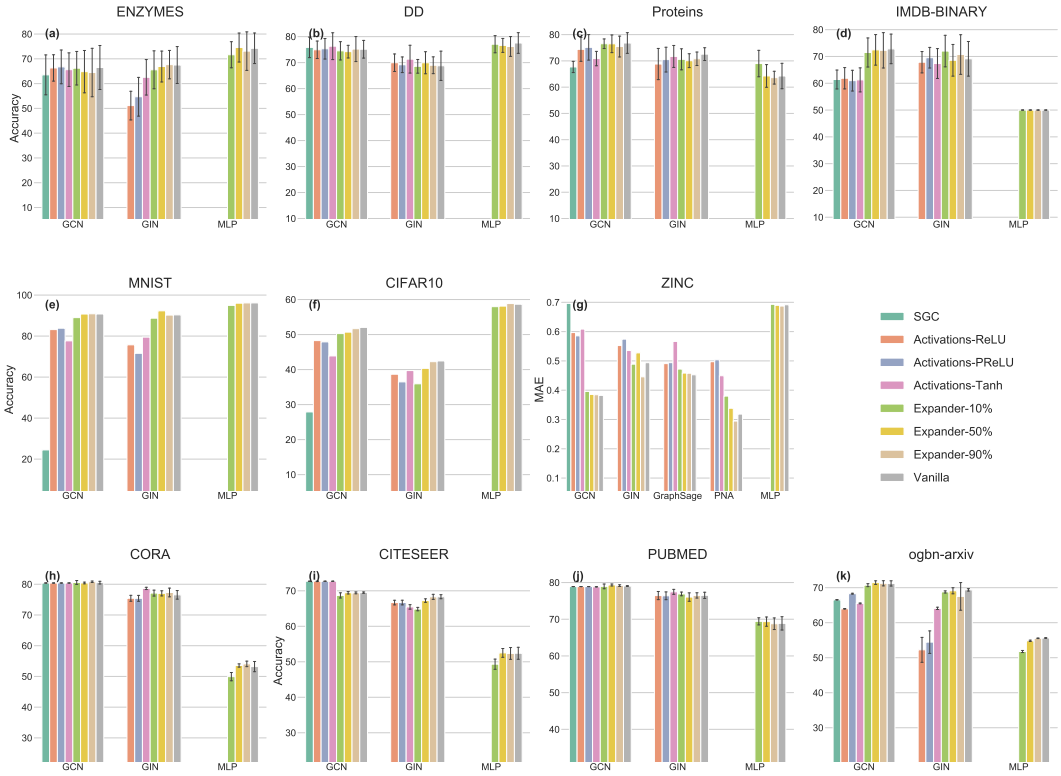

Figure 2: (a,b,c,d): Accuracy of different model types for GCN, GIN and MLP on ENZYMES/DD/PROTEINS/IMDB-BINARY; (e,f): Accuracy of different model types for GCN, GIN and MLP on MNIST/CIFAR10; (g): Mean Absolute Error of different model types for GCN/GIN/GraphSage/PNA/MLP on ZINC dataset; (h,i,j,k): Accuracy of different model types for GCN/GIN/MLP on CORA/CITESEER/PUBMED/ogbn-arxiv datasets.

4.3 Graph Regression

In Figure 2(g) the Mean Absolute Error (MAE) of our studied and proposed models on the ZINC dataset for graph regression is displayed. Similar to the graph classification task, the *Expander*GCN and *Expander*GraphSage models are on the same level with their corresponding vanilla models, regardless of their densities. The performance of the *Expander*GIN and *Expander*PNA models exhibits greater variance across the different densities, especially in the case of the PNA models the performance is increasing as the network gets denser indicating that the density of the *Update* step does positively contribute to the model performance of the PNA for the task of graph regression on the ZINC dataset. The *Activation-Only* models perform worse than their *Expander* counterparts on this task, again confirming the insight from the results of the *Expander*GNNs that the linear transform in the *Update* step does improve performance in this graph regression task. Again we see that *Activation-Only* GCNs outperform the SGC benchmark in this set of experiments.

Hence, for the task of graph regression we observe that both the linear transformation and non-linear activation function in the *Update* step have a positive impact on model performance. We might have been able to expect that the addition of the transformation performed in the *Update* step is of greater impact in a regression task, which is evaluated on a continuous scale, than in a classification task, where only a discrete label needs to be inferred.

4.4 Node Classification

Results from the node classification experiments on four citation graphs (CORA, CITESEER, PUBMED and ogbn-arxiv) can be found in Figure 2(h), (i), (j) and (k), respectively. For medium-sized datasets such as CORA, CITESEER and PUBMED, we have the same observation as for the graph classification and graph regression tasks discussed in Sections 4.2 and 4.3, the *Expander* models,

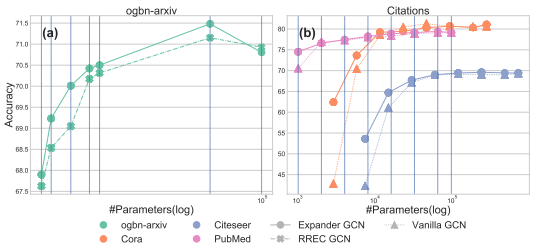

Figure 3: (a,b): Accuracy vs. Number of parameters plot on a logarithmic x-axis on (a) ogbn-arxiv for GCN models with different sparsifiers and (b) on CORA/CITESEER/PUBMED for vanilla and Expander GCN under the same parameter budget.

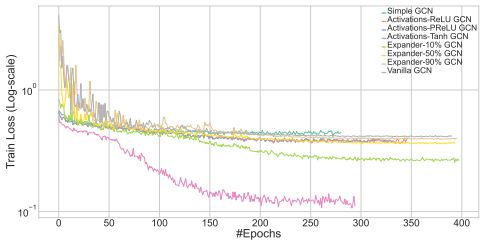

Figure 4: Training loss (cross-entropy) convergence behaviour of the different model types for the GCN used for graph classification on the PROTEINS dataset.

regardless of their sparsity, are performing on par with the vanilla ones. Only on the CITESEER dataset, we observe that the *Expander*GIN model with 10% density shows a small but non-negligible drop in performance compared to the vanilla model; while the 50% and 90% dense *Expander* models remain comparable to the vanilla one. The *Activation-Only* models also perform as well as or even better than (on CITESEER) the vanilla model. The performance of GCN *Activation-Only* model and SGC is equally good across all three datasets.

These conclusions remain true for large-scale datasets like ogbn-arxiv with 169,343 nodes and 1,166,243 edges. The *ExpanderGCNs* are on par with the vanilla GCN while the *Activation-Only* model and SGC perform slightly worse. However, the training time of *Activation-Only* model and SGC is five times faster than that of the *Expander* and vanilla models. The PReLU *Activation-Only* model performs better than the SGC, while the other two *Activation-Only* models do worse.

We observe that in the node classification task both the linear transformation and the non-linear activation function offer no benefit for the medium scale datasets. For the large-scale dataset we find that the linear transformation can be sparsified heavily without a loss in performance, but deleting it entirely does worsen model performance.

4.5 Expander Sparsification

In Figure 3(a) we compare the results from an *ExpanderGCN* to those achieved by a GCN model, where the expander linear layer, presented in Definition 1, is replaced by a deterministic sparsification construction from Bourely et al. (2017) the "Regular Rotating Edge Construction" (RREC). Bourely et al. (2017) observe that sparsifiers obtained from the RREC sampler have a significantly lower algebraic connectivity than the Expander Linear Layer sampler which we chose to utilise in the *Expander*GNNs. We observe that *Expander*GCNs outperform the RREC sampled GCN for almost all parameter budgets. Therefore, we confirm that the Expander Linear Layer is an appropriate choice for the *Expander*GNN model class.

The experiments in Sections 4.2, 4.3 and 4.4 show that the linear transform layer in the *Update* step of GNNs can often be sparsified to an arbitrary level without loss of performance. From this a natural question then arises: Will a shrunk model, i.e., a model with a smaller hidden dimension used in the *Update* step, matching the number of parameters of the sparsified *Expander*GNN, perform on par with its *Expander*GNN counterpart?

To study this question we compare the performance of vanilla GCNs to *Expander*GCNs with equally many parameters, but doubling the size of the hidden dimension of the vanilla GCN. Figure 3(b) shows the experiment results on the three citation datasets. We observe that for most parameter values the *Expander*GCN outperforms the vanilla GCN. This phenomenon becomes more evident when the number of parameter is small. In conclusion, it seems to be beneficial to choose a sparsified large model rather than a compact model with equally many parameters.

4.6 Convergence Behaviour

In Figure 4 we observe the training loss convergence of the vanilla GCN, *Expander*GCNs, *Activation-Only* GCNs and the SGC. When training these models we have implemented a learning rate decay scheme, where the training process is terminated if the learning rate drops below 10^{-6}. We are able to observe, that the number of epochs required for convergence is roughly equal for the *Activation-Only* GCN and SGC, as well as the *Expander*GCN and the vanilla GCN. For both pairwise comparisons we are able to observe, that our proposed models converge to a lower training loss than their counterparts.

5 Conclusion

With extensive experiments across different GNN models and graph learning tasks, we are able to confirm that the *Update* step can be sparsified heavily without a significant performance cost. In fact for seven of the eleven tested datasets across a variety of tasks we found that the linear transform can be removed entirely without a loss in performance, i.e., the *Activation-Only* models performed on par with their vanilla counterparts. The *Activation-Only* GCN model consistently outperformed the SGC model and especially in the computer vision datasets we witnessed that the activation functions seem to be crucial for good model performance accounting for an accuracy difference of up to 59%. These findings partially support the hypothesis by Wu et al. (2019) that the *Update* step can be simplified significantly without a loss in performance. Contrary to Wu et al. (2019) we find that the nonlinear activation functions result in a significant accuracy boost and the linear transformation in the *Update* step can be removed or heavily sparsified.

The *Activation-Only* GNN is an effective and simple benchmark model framework for any message passing neural network. It enables practitioners to test whether they can cut the large amount of model parameters used in the linear transform of the *Update* steps. If the linear transform does contribute positively to the model's performance then the *Expander*GNNs provide a model class of tuneable sparsity which allows efficient parameter usage.

References

Blalock, D., Ortiz, J. J. G., Frankle, J. & Guttag, J. (2020), What is the state of neural network pruning?, *in* 'Conference on Machine Learning and Systems'.

Bölcskei, H., Grohs, P., Kutyniok, G. & Petersen, P. (2019), 'Optimal approximation with sparsely connected deep neural networks', *SIAM Journal on Mathematics of Data Science* pp. 8–45.

Bourely, A., Boueri, J. P. & Choromonski, K. (2017), 'Sparse neural networks topologies', *arXiv:1706.05683* .

CEREBRAS SYSTEMS, I. (2021), 'Cerebras white paper 3: Cerebras systems: Achieving industry best ai performance through a systems approach', https://cerebras.net/wp-content/uploads/2021/04/Cerebras-CS-2-Whitepaper.pdf. accessed May 2021.

Chen, T., Bian, S. & Sun, Y. (2020), 'Are powerful graph neural nets necessary? a dissection on graph classification', *arXiv:1905.04579* .

Corso, G., Cavalleri, L., Beaini, D., Liò, P. & Veličković, P. (2020), 'Principal neighbourhood aggregation for graph nets', *arXiv:2004.05718* .

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A. & Adams, R. P. (2015), Convolutional networks on graphs for learning molecular fingerprints, *in* 'Advances in neural information processing systems', pp. 2224 – 2232.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y. & Bresson, X. (2020*a*), 'Benchmarking graph neural networks', *arXiv:2003.00982* .

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y. & Bresson, X. (2020*b*), 'Benchmarking graph neural networks', https://github.com/graphdeeplearning/benchmarking-gnns. accessed May 2020.

Frankle, J. & Carbin, M. (2019), The lottery ticket hypothesis: Finding sparse, trainable neural networks, *in* '7th International Conference on Learning Representations (ICLR)'.

Frankle, J., Dziugaite, G. K., Roy, D. M. & Carbin, M. (2021), Pruning neural networks at initialization: Why are we missing the mark?, *in* '9th International Conference on Learning Representations (ICLR)'.

Gama, F., Ribeiro, A. & Bruna, J. (2020), Stability of graph neural networks to relative perturbations, *in* 'International Conference on Acoustics, Speech, and Signal Processing (ICASSP)', IEEE, pp. 9070 – 9074.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. (2017), Neural message passing for quantum chemistry, *in* 'Proceedings of the 34th International Conference on Machine Learning (ICML)', pp. 1263 – 1272.

Griffa, A., Ricaud, B., Benzi, K., Bresson, X., Daducci, A., Vandergheynst, P., Thiran, J.-P. & Hagmann, P. (2017), 'Transient networks of spatio-temporal connectivity map communication pathways in brain functional systems', *NeuroImage* pp. 490 – 502.

Hamilton, W. L., Ying, R. & Leskovec, J. (2017), Inductive representation learning on large graphs, *in* 'Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)', Curran Associates Inc., pp. 1025 – 1035.

Han, S., Pool, J., Tran, J. & Dally, W. (2015), Learning both weights and connections for efficient neural network, *in* 'Advances in neural information processing systems', pp. 1135–1143.

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y. & Wang, M. (2020), 'Lightgcn: Simplifying and powering graph convolution network for recommendation', *arXiv preprint arXiv:2002.02126* .

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N. & Peste, A. (2021), 'Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks', *arXiv preprint arXiv:2102.00554* .

Hoory, S., Linial, N. & Wigderson, A. (2006), 'Expander graphs and their applications', *Bulletin of the American Mathematical Society* pp. 439 – 561.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M. & Leskovec, J. (2020*a*), 'Open graph benchmark: Datasets for machine learning on graphs', *arXiv preprint arXiv:2005.00687* .

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M. & Leskovec, J. (2020*b*), 'Open graph benchmark: Datasets for machine learning on graphs', https://github.com/snap-stanford/ogb. accessed October 2020.

Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S. & Coleman, R. G. (2012), 'Zinc: a free tool to discover chemistry for biology', *Journal of chemical information and modeling* pp. 1757 – 1768.

Kepner, J. & Robinett, R. (2019), Radix-net: Structured sparse matrices for deep neural networks, *in* '2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)', IEEE, pp. 268 – 274.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P. & Neumann, M. (2016), 'Benchmark data sets for graph kernels'. http://graphkernels.cs.tu-dortmund.de.

Kipf, T. N. & Welling, M. (2017), Semi-supervised classification with graph convolutional networks, *in* '5th International Conference on Learning Representations (ICLR)'.

Knyazev, B., Taylor, G. W. & Amer, M. (2019), Understanding attention and generalization in graph neural networks, *in* 'Advances in Neural Information Processing Systems 32', Curran Associates, Inc., pp. 4202 – 4212.

Lee, N., Ajanthan, T. & Torr, P. H. (2019), SNIP: Single-shot network pruning based on connection sensitivity, *in* '7th International Conference on Learning Representations (ICLR)'.

Lubotzky, A. (2012), 'Expander graphs in pure and applied mathematics', *Bulletin of the American Mathematical Society* pp. 113 – 162.

Luzhnica, E., Day, B. & Liò, P. (2019), On graph classification networks, datasets and baselines, *in* 'Workshop on Learning and Reasoning with Graph-Structured Data (ICML)'.

McDonald, A. W. & Shokoufandeh, A. (2019), Sparse super-regular networks, *in* '18th IEEE International Conference On Machine Learning And Applications (ICMLA)', IEEE, pp. 1764 – 1770.

Monti, F., Frasca, F., Eynard, D., Mannion, D. & Bronstein, M. M. (2019), 'Fake news detection on social media using geometric deep learning', *arXiv:1902.06673* .

NVIDIA (2020), 'Nvidia white paper: Nvidia a100 tensor core gpu architecture', https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf. accessed May 2021.

Prabhu, A., Varma, G. & Namboodiri, A. (2018), Deep expander networks: Efficient deep networks from graph theory, *in* 'Proceedings of the European Conference on Computer Vision (ECCV)', pp. 20 – 35.

Salha, G., Hennequin, R. & Vazirgiannis, M. (2019), Keep it simple: Graph autoencoders without graph convolutional networks, *in* 'Advances in Neural Information Processing Systems (NeurIPS)'.

Salha, G., Hennequin, R. & Vazirgiannis, M. (2020), Simple and effective graph autoencoders with one-hop linear models, *in* 'European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)'.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B. & Eliassi-Rad, T. (2008), 'Collective classification in network data', *AI magazine* pp. 93 – 93.

Strategy, M. I. . (2020), 'Graphcore white paper: The graphcore second generation ipu', https://www.graphcore.ai/hubfs/MK2-%20The%20Graphcore%202nd%20Generation%20IPU%20Final%20v7.14.2020.pdf?hsLang=en. accessed May 2021.

Tanaka, H., Kunin, D., Yamins, D. L. & Ganguli, S. (2020), 'Pruning neural networks without any data by iteratively conserving synaptic flow', *arXiv:2006.05467* .

Wang, C., Zhang, G. & Grosse, R. (2020), Picking winning tickets before training by preserving gradient flow, *in* '8th International Conference on Learning Representations (ICLR)'.

Wang, K., Shen, Z., Huang, C., Wu, C.-H., Dong, Y. & Kanakia, A. (2020), 'Microsoft academic graph: When experts are not enough', *Quantitative Science Studies* **1**(1), 396–413.

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J. & Zhang, Z. (2019), 'Deep graph library: A graph-centric, highly-performant package for graph neural networks', https://www.dgl.ai/. accessed April 2020.

Waradpande, V., Kudenko, D. & Khosla, M. (2020), 'Deep reinforcement learning with graph-based state representations', *arXiv preprint arXiv:2004.13965* .

Wu, F., Souza Jr, A. H., Zhang, T., Fifty, C., Yu, T. & Weinberger, K. Q. (2019), Simplifying graph convolutional networks, *in* 'Proceedings of the 36th International Conference on Machine Learning (ICML)'.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. & Philip, S. Y. (2020), 'A comprehensive survey on graph neural networks', *IEEE Transactions on Neural Networks and Learning Systems* pp. 1 – 21.

Xu, K., Hu, W., Leskovec, J. & Jegelka, S. (2019), How powerful are graph neural networks?, *in* '7th International Conference on Learning Representations (ICLR)'.

Yao, L., Mao, C. & Luo, Y. (2019), Graph convolutional networks for text classification, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', pp. 7370 – 7377.

A Full Experiment Results

Table A1: Results of the considered GNN models on different datasets

			ENZYMES			DD			PROTEINS			IMDB-BINARY		
			ACC	Time per Epoch(s)	#Parameters	ACC	Time per Epoch(s)	#Parameters	ACC	Time per Epoch(s)	#Parameters	ACC	Time per Epoch(s)	#Parameters
Simple	GCN	—	63.67 ± 8.06	0.2004	37853	75.90 ± 3.93	1.3841	43643	67.65 ± 2.21	0.3374	39311	61.30 ± 3.61	0.3297	31499
Activations	GCN	lelu	66.67 ± 6.24	0.2073	37853	74.79 ± 3.46	1.3979	43643	75.20 ± 5.19	0.3464	39311	61.30 ± 4.05	0.3418	31499
		prelu	66.67 ± 6.71	0.2486		75.39 ± 3.98	1.4643		75.29 ± 4.85	0.3654		61.00 ± 3.92	0.388	
		relu	66.33 ± 5.31	0.2132		74.88 ± 3.41	1.3915		74.48 ± 4.61	0.348		61.90 ± 4.01	0.3497	
		selu	66.17 ± 6.99	0.206		75.64 ± 4.54	1.3948		75.92 ± 2.88	0.3469		62.70 ± 3.32	0.3479	
		tanh	65.67 ± 7.04	0.2154		76.57 ± 5.20	1.391		70.89 ± 2.70	0.3452		61.20 ± 4.56	0.3522	
	GIN	lelu	52.17 ± 5.78	0.3219	5420	70.12 ± 4.30	1.6544	10610	67.74 ± 4.82	0.4978	4410	68.10 ± 4.50	0.5292	1282
		prelu	55.17 ± 7.94	0.3531		69.19 ± 3.02	1.7253		70.34 ± 4.78	0.5144		69.50 ± 3.88	0.577	
		relu	51.00 ± 5.88	0.3122		70.03 ± 3.27	1.6424		68.82 ± 5.97	0.4897		67.90 ± 3.96	0.5262	
		selu	53.83 ± 7.34	0.3201		72.49 ± 4.30	1.6772		72.40 ± 5.03	0.4989		69.00 ± 6.03	0.5345	
		tanh	62.83 ± 7.15	0.3073		71.32 ± 5.29	1.6443		71.96 ± 4.26	0.494		67.30 ± 5.62	0.5233	
Expander	GCN	10%	66.33 ± 6.78	0.2863	22775	74.53 ± 3.50	1.5731	21064	76.55 ± 1.90	0.4524	22781	71.60 ± 5.50	0.4673	19920
		50%	64.83 ± 8.64	0.285	58293	74.28 ± 2.52	1.5856	56960	76.36 ± 3.43	0.448	58948	72.40 ± 5.70	0.4779	50888
		90%	64.83 ± 9.44	0.2917	93209	75.13 ± 4.69	1.5763	92215	75.38 ± 4.01	0.4476	94502	72.30 ± 6.65	0.4835	81303
	GIN	10%	65.83 ± 7.75	0.3798	8918	68.59 ± 2.70	1.6605	6730	70.53 ± 3.96	0.5481	6819	72.00 ± 6.16	0.6368	5850
		50%	67.00 ± 6.05	0.3822	29070	70.03 ± 4.20	1.6427	28789	70.08 ± 2.69	0.5483	27455	68.80 ± 5.90	0.6268	24125
		90%	67.50 ± 5.74	0.3729	49222	68.93 ± 3.26	1.637	50335	70.71 ± 2.55	0.5455	48091	70.30 ± 7.39	0.6127	41975
	MLP	10%	71.50 ± 5.13	0.1653	26001	77.00 ± 3.39	1.0672	23858	69.00 ± 4.99	0.2856	25453	50.00 ± 0.00	0.287	22538
		50%	74.67 ± 5.72	0.1783	59661	76.66 ± 2.69	1.07	58020	64.15 ± 4.32	0.2852	58928	50.00 ± 0.00	0.2896	51244
		90%	73.17 ± 7.65	0.1772	92811	76.24 ± 3.80	1.0621	91631	63.61 ± 2.40	0.2831	91888	50.00 ± 0.00	0.2967	79487
Vanilla	GCN	—	66.50 ± 8.71	0.2557	102239	75.13 ± 3.44	1.5782	101189	76.73 ± 3.85	0.4388	103697	72.70 ± 5.68	0.4433	89045
	GIN	—	67.67 ± 7.68	0.3692	54260	68.76 ± 5.55	1.6572	55978	72.51 ± 2.39	0.5309	53250	69.00 ± 6.23	0.6135	46650
	MLP	—	74.17 ± 6.34	0.1646	101481	77.43 ± 3.98	1.0523	100447	64.33 ± 4.93	0.2769	100643	50.00 ± 0.00	0.2725	86895

			CORA			CITESEER			PUBMED			ogbn-arxiv		
			ACC	Time per Epoch(s)	#Parameters	ACC	Time per Epoch(s)	#Parameters	ACC	Time per Epoch(s)	#Parameters	ACC	Time per Epoch(s)	#Parameters
Simple	GCN	—	80.40 ± 0.00	0.0088	10038	72.70 ± 0.00	0.0116	22224	78.90 ± 0.00	0.0148	1503	66.53 ± 0.07	0.0501	5160
Activations	GCN	prelu	80.40 ± 0.00	0.0112	10038	72.70 ± 0.00	0.0159	22224	78.90 ± 0.00	0.0201	1503	68.29 ± 0.13	0.0603	5160
		relu	80.40 ± 0.00	0.0088		72.70 ± 0.00	0.0124		78.90 ± 0.00	0.0156		63.97 ± 0.09	0.0519	
		tanh	80.40 ± 0.00	0.0086		72.70 ± 0.00	0.0125		78.83 ± 0.05	0.0157		65.51 ± 0.12	0.0518	
	GIN	prelu	75.42 ± 0.97	0.0124	30114	66.70 ± 0.65	0.0211	66672	76.38 ± 1.06	0.0245	4509	54.45 ± 3.20	0.0732	15480
		relu	75.42 ± 0.97	0.0117		66.70 ± 0.65	0.0202		76.46 ± 1.12	0.0235		52.21 ± 3.47	0.071	
		tanh	78.65 ± 0.36	0.0121		65.45 ± 0.63	0.0202		77.46 ± 0.67	0.0236		64.10 ± 0.32	0.0711	
Expander	GCN	10%	80.59 ± 0.64	0.021	38663	68.68 ± 0.73	0.0244	96518	78.95 ± 0.63	0.0237	13827	70.70 ± 0.42	0.2733	20392
		50%	80.42 ± 0.28	0.0228	185351	69.43 ± 0.34	0.0278	475654	79.34 ± 0.28	0.0246	65027	71.42 ± 0.55	0.2681	59944
		90%	80.82 ± 0.23	0.0213	332039	69.42 ± 0.26	0.0259	854790	79.17 ± 0.23	0.0236	116227	71.22 ± 0.71	0.2732	99112
	GIN	10%	77.08 ± 0.96	0.0163	57156	64.83 ± 0.49	0.022	126940	76.91 ± 0.51	0.0235	22757	68.78 ± 0.31	0.2742	52768
		50%	77.06 ± 0.81	0.017	230212	67.24 ± 0.49	0.0253	532444	76.06 ± 1.14	0.024	100325	69.11 ± 0.86	0.2742	118688
		90%	77.36 ± 1.40	0.0167	403012	68.31 ± 0.70	0.0229	937692	76.50 ± 0.73	0.0236	177637	67.54 ± 3.91	0.274	183968
	MLP	10%	49.91 ± 1.34	0.0096	38663	49.34 ± 1.43	0.0143	96518	69.43 ± 0.94	0.0142	13827	51.77 ± 0.30	0.1109	20392
		50%	53.57 ± 0.53	0.0111	185351	52.53 ± 1.25	0.0175	475654	69.34 ± 1.24	0.0151	65027	54.83 ± 0.20	0.1106	59944
		90%	54.06 ± 0.88	0.0102	332039	52.34 ± 1.64	0.0154	854790	68.78 ± 1.56	0.0145	116227	55.55 ± 0.08	0.1107	99112
Vanilla	GCN	—	80.54 ± 0.44	0.0172	368903	69.50 ± 0.19	0.0194	949766	79.04 ± 0.12	0.0215	129027	71.22 ± 0.76	0.2713	109096
	GIN	—	76.57 ± 1.36	0.0126	446532	68.33 ± 0.56	0.0154	1039324	76.55 ± 0.84	0.022	197093	69.37 ± 0.34	0.2737	200608
	MLP	—	53.21 ± 1.69	0.0061	368903	52.44 ± 1.71	0.0086	949766	68.87 ± 1.78	0.0129	129027	55.63 ± 0.11	0.1098	109096

			MNIST			CIFAR10			ZINC		
			ACC	Time per Epoch(s)	#Parameters	ACC	Time per Epoch(s)	#Parameters	MAE	Time per Epoch(s)	#Parameters
Simple	GCN	—	24.48	100.098	35811	27.90	143.721	36103	0.6963	1.9847	34347
Activations	GCN	lelu	83.52	101.554	14349	48.31	39.6657	14641	0.5947	1.8624	13177
		prelu	83.84	103.11		47.90	41.3494		0.5855	1.9273	
		relu	83.16	102.443		48.27	40.2591		0.5967	1.8407	
		tanh	77.67	102.975		43.89	40.6699		0.6086	1.8639	
	GIN	lelu	75.68	119.803	5990	37.90	44.6216	6210	0.5368	2.8046	555
		prelu	71.60	115.862		36.48	46.0211		0.5743	2.8826	
		relu	75.73	119.081		38.67	45.0776		0.5524	2.8009	
		tanh	79.49	119.729		39.71	45.0382		0.5354	2.8263	
Expander	GCN	10%	89.00	124.411	22713	50.27	45.0057	22741	0.3958	2.5833	21877
		50%	90.75	124.075	57346	50.69	45.1236	57492	0.3856	2.5793	55517
		90%	90.87	124.166	91392	51.68	44.769	91654	0.3845	2.5578	89157
	GIN	10%	88.73	120.735	10973	35.93	44.4764	10995	0.4888	3.0905	5835
		50%	92.31	119.554	30465	40.35	44.537	30575	0.5274	3.1125	25195
		90%	90.24	116.378	49957	42.25	44.401	50155	0.4456	3.0852	44555
	MLP	10%	94.97	67.2461	26980	57.96	31.6988	27012	0.6931	2.1455	23775
		50%	96.04	66.9236	61456	58.12	31.6178	61624	0.6898	2.1426	59775
		90%	96.17	66.7391	95425	58.85	31.6307	95727	0.6873	2.1542	95775
Vanilla	GCN	—	90.77	124.091	100197	52.04	44.337	100489	0.3823	2.5538	97857
	GIN	—	90.33	120.191	54830	42.46	45.2386	55050	0.4939	3.0569	49395
	MLP	—	96.17	66.125	104044	58.66	31.5435	104380	0.6916	2.1085	104775

Remarks

1. We report in this table the full set of experiment results (accuracy/mean absolute error, training time per epoch and number of model parameters) we have performed, including TU datasets (ENZYMES, DD, PROTEINS and IMDB-BINARY), Citation Networks datasets (CORA, CITESEER, PUBMED and Arxiv), MNIST and CIFAR10, as well as ZINC.
2. All datasets can be downloaded from (and are processed following) Benchmarking Graph Neural Networks repository Dwivedi et al. (2020*b*), except the ogbn-arxiv dataset, which is downloadable from Open Graph Benchmark repository Hu et al. (2020*b*), as well as three other citations datasets downloadable from Deep Graph Library Wang et al. (2019).
3. We implement 10-fold cross validation on TU datasets and repeat the experiments 10 times for Citation Networks datasets, in order to eliminate impacts of randomness. We report their means and standard deviations in the form of $\mu \pm \sigma$.
4. In light of computational time and their considerably large sizes, we experiment only once on MNIST, CIFAR10 and ZINC.
5. Apart from the three choices of activation functions we report in the main paper (prelu, relu and tanh), we have also experimented on other activation functions. They are omitted in the main paper as their performances are very similar to those we choose. We keep their results in this table for reference.
6. Considering the computational cost of PNA and GraphSage, we only include them in experiments on ZINC. The results are summarized in the bottom table, where softsh stands for "softshrink" activation and GSage for "GraphSage".

		Activations																Expander						Vanilla	
		GCN		GIN		GSage						PNA						GSage			PNA			GSage	PNA
		selu	softsh	selu	softsh	lelu	prelu	relu	selu	softsh	tanh	lelu	prelu	relu	selu	softsh	tanh	10%	50%	90%	10%	50%	90%	—	—
ZINC (Extra results)	MAE	0.6128	0.6549	0.5424	0.5221	0.4937	0.494	0.4907	0.5365	1.5508	0.5665	0.5181	0.5038	0.4972	0.5285	0.5374	0.4493	0.4721	0.4584	0.4579	0.3798	0.3384	0.2946	0.4526	0.3184
	Time per Epoch(s)	1.8581	1.8574	2.8144	2.821	4.1406	4.3778	4.1238	4.1293	4.1428	4.121	27.9922	28.1269	28.0074	27.9645	28.0993	27.9598	4.461	4.4495	4.4581	32.0922	32.148	32.0666	4.4149	31.6077
	#Parameters	13177		555		5130						3515						11970	37890	63810	23735	102495	181255	70290	201205