SPARSIFYING THE UPDATE STEP IN GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

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, empirical 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.

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 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 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.

In this paper we empirically 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 Simple Graph Convolution model (SGC, Wu et al., 2019) where the activation functions where removed to merge the linear layers.

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.

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, well-performing CNN architecture they call X-nets. 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.

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 et al., 2020a; 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 Graph Convolutional Network (GCN, Kipf & Welling, 2017) 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. 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 general model structure of MPNNs and our two proposed model classes, where we sparsify or remove the linear transform layer in the *Update* step. Theoretical motivation and implementation details of our proposed models can be found in Appendix Section A.

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 $\boldsymbol{X} \in \mathbb{R}^{n \times s}$. The MPNN structure is a prominent paradigm for performing machine learning tasks on graphs such as node or graph classification. 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 $\boldsymbol{h}_i^{(1)}$ denotes the hidden representation of node i.

Aggregation. For each node information is gathered 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 equi- 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 neural network learning the update function at iteration l. Then, for node i, we have

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

Expander Update Step. If we denote the two sets of computational units, i.e., neurons, being connected in a fully-connected layer of a neural network by $\mathbb{S}_1$ and $\mathbb{S}_2$, respectively. Then, following Prabhu et al. (2018), we propose to sparsify this layer by uniform randomly sampling only d connections for the computational units in the smaller of the two connected sets $\mathbb{S}_1$ and $\mathbb{S}_2$. We refer to the *density* of such an expander linear layer as the ratio of the number of sampled connections to the number of connections in the fully-connected layer, i.e., $d/\max(|\mathbb{S}_1|, |\mathbb{S}_2|)$. 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. When we replace all linear layers in the *Update* steps of a GNN with expander linear layers, we get the *Expander*GNN.

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).

Activation-Only Models. In the *Activation-Only* GNN models, we propose to fully remove the linear transformation in the *Update* step 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.

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

4 Experiments and Discussion

In order to investigate the influence of the *Update* step in GNNs, we now observe the performance of the proposed benchmark models on the tasks of graph classification and regression. In the Appendix Section B we provide an overview of our experimentation setup. Additional experiments on node classification, confirming the trends observed in graph classification and regression, can be found in Appendix Section C. We make our anonymised experimentation code accessible online[1].

Graph Classification. Table 1 shows the experiment results of the vanilla GCN model and its *Expander* and *Activation-Only* variants on the ENZYMES, PROTEINS, MNIST and CIFAR datasets for graph classification. One direct observation is that the *Expander*GCN 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 and PROTEINS datasets. The SGC performs, often significantly, worse than the *Activation-Only* model, especially on the computer vision MNIST and CIFAR datasets. Additional experiments on the GIN architecture can be found in Appendix Table 7. They reaffirm the conclusions drawn on the GCN model on graph classification in this section.

[1] `https://www.dropbox.com/sh/hzb1x8bv6o6t1o6/AAAkjANtZy_1nJ2jGqvAMkQ3a`

Table 1: 10-fold Cross Validation results (mean $\pm$ std) of the accuracy of the GCN on the graph classification task performed on the ENZYMES/PROTEINS/MNIST/CIFAR10 datasets. We also report the number of parameters relative to the number of parameters in the vanilla model, e.g., 0.37 means that the number of parameters in *Activation-Only* model is 37% of the vanilla one. We set the best results to bold and underline the second best result. In addition, if the result of *Activation-Only* model is better than the SGC model, we put $*$ next to the result.

	ENZYMES		Proteins		MNIST		CIFAR10	
	ACC.	Params.	ACC.	Params.	ACC.	Params.	ACC.	Params.
Vanilla	$\underline{66.50 \pm 8.71}$	1.00	$\mathbf{76.73 \pm 3.85}$	1.00	$\mathbf{90.77}$	1.00	$\mathbf{52.04}$	1.00
SGC	63.67 ± 8.06	0.37	67.65 ± 2.21	0.38	24.48	0.36	27.90	0.36
Expander-50%	64.83 ± 8.64	0.57	76.36 ± 3.43	0.57	$\underline{90.75}$	0.57	$\underline{50.69}$	0.57
Expander-10%	66.33 ± 6.78	0.22	$\underline{76.55 \pm 1.90}$	0.22	89.00	0.23	50.27	0.23
Activation-Only	$\mathbf{66.67 \pm 6.71}*$	0.37	$75.92 \pm 2.88*$	0.38	$83.84*$	0.36	$48.31*$	0.36

Table 2: Results of the GCN/GIN/GraphSage/PNA models on graph regression for the ZINC dataset. The format follows Table 1.

	GCN		GIN		GraphSage		PNA	
	MAE	Params.	MAE	Params.	MAE	Params.	MAE	Params.
Vanilla	$\mathbf{0.3823}$	1.00	$\underline{0.4939}$	1.00	$\mathbf{0.4530}$	1.00	$\mathbf{0.3180}$	1.00
SGC	0.6963	0.35	—	—	—	—	—	—
Expander-50%	$\underline{0.3856}$	0.57	0.5274	0.51	$\underline{0.4580}$	0.54	$\underline{0.3380}$	0.51
Expander-10%	0.3958	0.22	$\mathbf{0.4888}$	0.12	0.4720	0.17	0.3800	0.12
Activation-Only	$0.5855*$	0.13	0.5220	0.01	0.4910	0.07	0.4490	0.02

Graph Regression. In Table 2 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 match the performance of 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 increases 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, 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 small, positive impact on model performance.

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 six of the nine 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.

REFERENCES

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? In *Conference on Machine Learning and Systems*, 2020.

Helmut Bölcskei, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. Optimal approximation with sparsely connected deep neural networks. *SIAM Journal on Mathematics of Data Science*, pp. 8–45, 2019.

Alfred Bourely, John Patrick Boueri, and Krzysztof Choromonski. Sparse neural networks topologies. *arXiv:1706.05683*, 2017.

INC CEREBRAS SYSTEMS. 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`, 2021. accessed May 2021.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv:2004.05718*, 2020.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224 – 2232, 2015.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv:2003.00982*, 2020a.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. `https://github.com/graphdeeplearning/benchmarking-gnns`, 2020b. accessed May 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations (ICLR)*, 2019.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *9th International Conference on Learning Representations (ICLR)*, 2021.

Fernando Gama, Alejandro Ribeiro, and Joan Bruna. Stability of graph neural networks to relative perturbations. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 9070 – 9074. IEEE, 2020.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 1263 – 1272, 2017.

Alessandra Griffa, Benjamin Ricaud, Kirell Benzi, Xavier Bresson, Alessandro Daducci, Pierre Vandergheynst, Jean-Philippe Thiran, and Patric Hagmann. Transient networks of spatio-temporal connectivity map communication pathways in brain functional systems. *NeuroImage*, pp. 490 – 502, 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, pp. 1025 – 1035. Curran Associates Inc., 2017.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, pp. 439 – 561, 2006.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, pp. 1757 – 1768, 2012.

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

Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. `http://graphkernels.cs.tu-dortmund.de`.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR)*, 2017.

Boris Knyazev, Graham W Taylor, and Mohamed Amer. Understanding attention and generalization in graph neural networks. In *Advances in Neural Information Processing Systems 32*, pp. 4202 – 4212. Curran Associates, Inc., 2019.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *7th International Conference on Learning Representations (ICLR)*, 2019.

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

Andrew WE McDonald and Ali Shokoufandeh. Sparse super-regular networks. In *18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pp. 1764 – 1770. IEEE, 2019.

Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *arXiv:1902.06673*, 2019.

NVIDIA. 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`, 2020. accessed May 2021.

Ameya Prabhu, Girish Varma, and Anoop Namboodiri. Deep expander networks: Efficient deep networks from graph theory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 20 – 35, 2018.

Guillaume Salha, Romain Hennequin, and Michalis Vazirgiannis. Keep it simple: Graph autoencoders without graph convolutional networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, pp. 93 – 93, 2008.

Moor Insights & Strategy. 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`, 2020. accessed May 2021.

Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv:2006.05467*, 2020.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *8th International Conference on Learning Representations (ICLR)*, 2020a.

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020b.

Felix Wu, Amauri H Souza Jr, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1 – 21, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations (ICLR)*, 2019.

Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7370 – 7377, 2019.

APPENDIX

A MOTIVATION AND IMPLEMENTATION OF OUR PROPOSED MODELS

Motivation of the Expander Linear Layer From our random expander sampling scheme discussed in Section 3 we sample 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 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. To illustrate this alteration we use the following matrix representation of the GCN's model equation,

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

In our implementation the *Expander*GCN has the following model equation,

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

Motivation of the Activation-Only Models. In Gama et al. (2020) it is argued that the nonlinearity 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.

B 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% and 50%), the *Activation-Only* GNN model

Table 3: Model Equations of the Vanilla, *Expander* and *Activation-Only* GNNs.

Model		Aggregation	Update
GCN	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = \frac{1}{\sqrt{d_i}} \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)} \frac{1}{\sqrt{d_j}}$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$
	Activation-Only		$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{m}_i^{(l)}\right)$
GIN	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = (1+\epsilon)\boldsymbol{h}_i^{(l)} + \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$
	Activation-Only		$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{m}_i^{(l)}\right)$
GraphSage	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = \mathrm{CONCAT}\left(\boldsymbol{h}_i^{(l)}, \mathrm{MAX}_{j \in \mathcal{N}(i)} \sigma(\boldsymbol{h}_j^{(l)} \boldsymbol{M}_1^{(l)} \odot \boldsymbol{W}_1^{(l)})\right)$	$\boldsymbol{h}_i^{(l+1)} = \frac{\sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}_2^{(l)} \odot \boldsymbol{W}_2^{(l)})}{\lVert \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}_2^{(l)} \odot \boldsymbol{W}_2^{(l)}) \rVert_2}$
	Activation-Only	$\boldsymbol{m}_i^{(l)} = \boldsymbol{h}_i^{(l)} + \mathrm{MAX}_{j \in \mathcal{N}(i)} \sigma(\boldsymbol{h}_j^{(l)})$	$\boldsymbol{h}_i^{(l+1)} = \frac{\sigma(\boldsymbol{m}_i^{(l)})}{\lVert \sigma(\boldsymbol{m}_i^{(l)}) \rVert_2}$
PNA	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = \mathrm{CONCAT}_{s,a}\left(\bigoplus_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}\right)$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$
	Activation-Only	$\boldsymbol{m}_i^{(l)} = \frac{1}{12} \sum_{s,a} \left[\bigoplus_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}\right]_{i,s,a}$	$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{m}_i^{(l)}\right)$

(we report the best result obtained from the ReLU, PReLU and Tanh activation functions), 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)). The precise model equations of our proposed architectures applied to these GNNs can be found in Table 3. We make the following additional remarks with regard to this table:

1. In the model equations of the GCN, d_i denotes the degree of node i.
2. In the model equations of the GIN, ϵ is a learnable ratio added explicitly to the central node's own representation.
3. In the model equations of the 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).

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 two TU datasets (Kersting et al., 2016) which are chemical 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., 2020a), 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. (2020a). 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 et al., 2020b; Hu et al., 2020), where the nodes are academic articles linked by citations. Details of the used datasets can be found in Table 4, 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. (2020a). Specifically, we follow the same training procedure, such as train/valid/test dataset splits, choice of optimiser, learning rate decay scheme, as well as the same hyperparameters, 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. (2020b). 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

Table 4: Properties of all datasets used in experiments.

	Dataset	#Graphs	#Nodes (avg.)	#Edges (avg.)	Task
TU datasets	ENZYMES	600	32.63	62.14	Graph Classification
	PROTEINS	1113	39.06	72.82	
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	

Table 5: 10-fold Cross Validation results (mean $\pm$ std) of the accuracy of the GCN on node classification for the CORA/CiteSeer/PubMed/OGBN-Arxiv datasets. The format follows Table 1 in the main text.

	Cora		CiteSeer		PubMed		OGBN-Arxiv	
	ACC.	Params.	ACC.	Params.	ACC.	Params.	ACC.	Params.
Vanilla	80.54 ± 0.44	1.00	$\underline{69.50 \pm 0.19}$	1.00	$\underline{79.04 \pm 0.12}$	1.00	$\underline{71.22 \pm 0.76}$	1.00
SGC	80.40 ± 0.00	0.03	$\mathbf{72.70 \pm 0.00}$	0.02	78.90 ± 0.00	0.01	66.53 ± 0.07	0.05
Expander-50%	$\underline{80.42 \pm 0.28}$	0.50	69.43 ± 0.34	0.50	$\mathbf{79.34 \pm 0.28}$	0.50	$\mathbf{71.42 \pm 0.55}$	0.56
Expander-10%	$\mathbf{80.59 \pm 0.64}$	0.10	68.68 ± 0.73	0.10	78.95 ± 0.63	0.11	70.70 ± 0.42	0.19
Activation-Only	80.40 ± 0.00	0.03	$\mathbf{72.70 \pm 0.00}$	0.02	78.90 ± 0.00	0.01	68.29 ± 0.13	0.05

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.

C Node Classification Results

Results from the node classification experiments on four citation graphs (CORA, CITESEER, PUBMED and ogbn-arxiv) can be found in Table 5. For medium-sized datasets such as CORA, CITESEER and PUBMED, we have the same observation as for the graph classification and graph regression tasks, the *Expander* models, regardless of their sparsity, are performing on par with the

Table 6: 10-fold Cross Validation Results (mean $\pm$ std) of the accuracy of the GIN on node classification for Cora/CiteSeer/PubMed/OGBN-Arxiv. The format follows Table 1 in the main text.

	Cora		CiteSeer		PubMed		OGBN-Arxiv	
	ACC.	Params.	ACC.	Params.	ACC.	Params.	ACC.	Params.
Vanilla	76.57 ± 1.36	1.00	$\mathbf{68.33 \pm 0.56}$	1.00	76.55 ± 0.84	1.00	$\mathbf{69.37 \pm 0.34}$	1.00
Expander-50%	77.06 ± 0.81	0.52	67.24 ± 0.49	0.51	76.06 ± 1.14	0.51	69.11 ± 0.86	0.59
Expander-10%	77.08 ± 0.96	0.13	64.83 ± 0.49	0.12	76.91 ± 0.51	0.12	68.78 ± 0.31	0.26
Activation-Only	$\mathbf{78.65 \pm 0.36}$	0.07	66.70 ± 0.65	0.06	$\mathbf{77.46 \pm 0.67}$	0.02	64.10 ± 0.32	0.08

Table 7: 10-fold Cross Validation results (mean $\pm$ std) of the accuracy of the GIN on Graph classification for ENZYMES and Proteins, as well as Results for MNIST and CIFAR10. The format follows Table 1 in the main text.

	ENZYMES		Proteins		MNIST		CIFAR10	
	ACC.	Params.	ACC.	Params.	ACC.	Params.	ACC.	Params.
Vanilla	$\mathbf{67.67 \pm 7.68}$	1.00	$\mathbf{72.51 \pm 2.39}$	1.00	90.33	1.00	**42.46**	1.00
Expander-50%	67.00 ± 6.05	0.54	70.08 ± 2.69	0.52	**92.31**	0.56	40.35	0.56
Expander-10%	65.83 ± 7.75	0.16	70.53 ± 3.96	0.13	88.73	0.20	35.93	0.20
Activation-Only	62.83 ± 7.15	0.10	72.40 ± 5.03	0.08	79.49	0.11	39.71	0.11

vanilla ones. The *Activation-Only* models also perform as well as or even better than (on CITESEER) the vanilla model. The performance of the GCN *Activation-Only* model and SGC is equally good across all three datasets.

These conclusions remain true for the large-scale dataset 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 *Activation-Only* model outperforms the SGC.

These results are further supported by the findings on the GIN architecture displayed in Table 6.

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.