DON'T STACK LAYERS IN GRAPH NEURAL NETWORKS, WIRE THEM RANDOMLY

Anonymous authors
Paper under double-blind review

ABSTRACT

Several results suggest an inherent difficulty of graph neural networks in extracting better performance by increasing the number of layers. Recent works attribute this to a phenomenon peculiar to the extraction of node features in graph-based tasks, i.e., the need to consider multiple neighborhood sizes at the same time and adaptively tune them. In this paper, we investigate the recently proposed randomly wired architectures in the context of graph neural networks. Instead of building deeper networks by stacking many layers, we prove that employing a randomly-wired architecture can be a more effective way to increase the capacity of the network and obtain richer representations. We show that such architectures behave like an ensemble of paths, which are able to merge contributions from receptive fields of varied size. Moreover, these receptive fields can also be modulated to be wider or narrower through the trainable weights over the paths.

1 INTRODUCTION

Several works, starting with the early GCN (Kipf & Welling, 2017), noticed an inability to build deep graph neural networks (GNNs). When trying to build anything but very shallow networks, GNNs often result in worse performance than that of methods that disregard the graph domain. This calls for exploring whether advances on CNN architectures can be translated to the GNN space, while understanding the potentially different needs of graph representation learning.

Li et al. (2019) suggest that GCNs suffer from oversmoothing as several layers are stacked, resulting in the extraction of mostly low-frequency features. This is related to the lack of self-loop information in this specific graph convolution. It is suggested that ResNet-like architectures mitigate the problem as the skip connections supply high frequency contributions. Xu et al. (2018) point out that the size of the receptive field of a node, i.e., which nodes contribute to the features of the node under consideration, plays a crucial role, but it can vary widely depending on the graph and too large receptive fields may actually harm performance. They conclude that for graph-based problems it would be optimal to learn how to adaptively merge contributions from receptive fields of multiple size. For this reason they propose an architecture where each layer has a skip connection to the output so that contributions at multiple depths (hence sizes of receptive fields) can be merged. Nonetheless, the problem of finding methods for effectively increasing the capacity of graph neural networks is still standing, since stacking many layers has been proven to provide limited improvements (Li et al., 2019; Oono & Suzuki, 2019; Alon & Yahav, 2020; NT & Maehara, 2019).

In this paper, we argue that the recently proposed randomly wired architectures (Xie et al., 2019) are ideal for GNNs. In a randomly wired architecture, "layers" are arranged according to a random directed acyclic graph and data are propagated through the paths towards the output. Such architecture is ideal for GNNs because it realizes the intuition of Xu et al. (2018) of being able of merging receptive fields of varied size. Indeed, the randomly wired network can be seen as a generalization of their approach where layer outputs can not only jump to the network output but to other layers as well, continuously merging receptive fields. Hence, randomly wired architectures provide a way of effectively scaling up GNNs, mitigating the depth problem and creating richer representations. Fig. 1 shows a graphical representation of this concept by highlighting the six layers directly contributing to the output, having different receptive fields induced by the distribution of paths from the input.

2 RANDOMLY WIRED GNNS

In recent work, Xie et al. (2019) explore whether it is possible to avoid handcrafted design of neural network architectures and, at the same time, avoid expensive neural architecture search methods

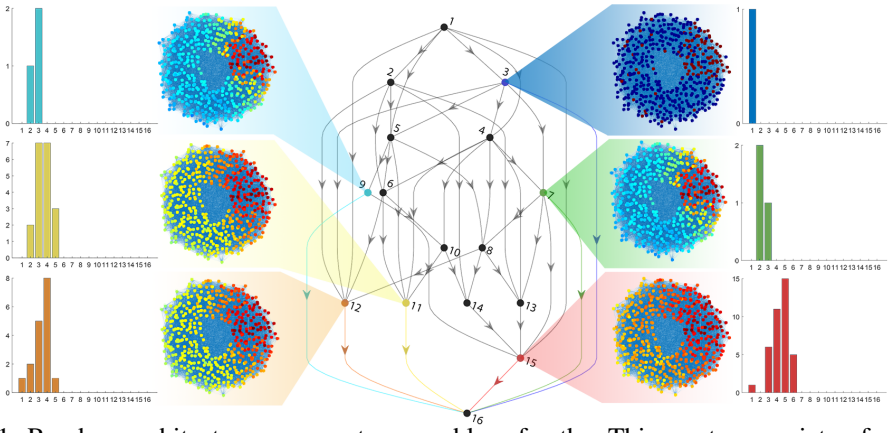

Figure 1: Random architectures aggregate ensembles of paths. This creates a variety of receptive fields (effective neighborhood sizes on the domain graph) that are combined to compute the output. Figure shows the domain graph where nodes are colored (red means high weight, blue low weight) according to the receptive field weighted by the path distribution of a domain node. The receptive field is shown at all the architecture nodes directly contributing to the output. Histograms represent the distribution of path lengths from source to architecture node.

(Elsken et al., 2019), by designing random architecture generators. They show that "layers" performing convolution, normalization and non-linear activation can be connected in a random architecture graph. Strong performance is observed on the traditional image classification task by outperforming state-of-the-art architectures. The authors conjecture that random architectures generalize ResNets and similar constructions, but the underlying principles of their excellent performance are unclear, as well as whether the performance translates to tasks other than image recognition or to operations other than convolution on grids.

A randomly wired architecture consists of a directed acyclic graph (DAG) connecting a source architecture node, which is fed with the input data, to a sink architecture node. One should not confuse the architecture DAG with the graph representing the GNN domain: to avoid any source of confusion we will use the terms *architecture nodes* (edges) and *domain nodes* (edges), respectively. A domain node is a node of the graph that is fed as input to the GNN. An architecture node is effectively a GNN layer performing the following operations (Fig. 2): i) aggregation of the inputs from other architecture nodes via a weighted sum as in (Xie et al., 2019):

$$\mathbf{h}^{(i)} = \sum_{j \in \mathcal{A}_i} \omega_{ij} \mathbf{h}^{(j)} = \sum_{j \in \mathcal{A}_i} \sigma(w_{ij}) \mathbf{h}^{(j)}, \quad i = 1, ..., L-1 \quad (1)$$

being σ a sigmoid function, $\mathbf{h}^{(i)}$ the output feature of the i-th node, $\mathcal{A}_i$ the set of direct predecessors of the architecture node i, and w_{ij} a scalar trainable weight; ii) a non-linear activation; iii) a graph-convolution operation (without output activation); iv) batch normalization.

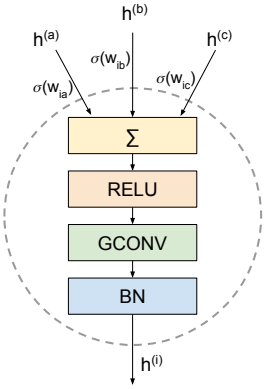

Figure 2: An architecture node is equivalent to a GNN layer.

The architecture DAG is generated using a random graph generator. In this paper, we focus on the Erdős-Renyi model. If multiple input architecture nodes are randomly generated, they are all wired to a single global input. Multiple output architecture nodes are averaged to obtain a global output.

2.1 Randomly wired architectures behave like path ensembles

It has already been shown that ResNets behave like ensembles of relatively shallow networks, where one can see the ResNet architecture as a collection of paths of varied lengths (Veit et al., 2016). More specifically, in a ResNet with n layers, where all layers have a skip connection except the first one and the last one, there are exactly 2^{L-2} paths, whose lengths follow a Binomial distribution (i.e., the number of paths of length l from layer k to the last layer is $\binom{L-k-1}{l-2}$), and the average path length is $\frac{L}{2} + 1$ (Veit et al., 2016). A randomly wired neural network can also be considered as an ensemble of networks with varied depth. However, in this case, the distribution of the path length is different from the one obtained with the ResNet, as shown in the following lemma.

Lemma 2.1. *Let us consider a randomly wired network with L architecture nodes, where the architecture DAG is generated according to a Erdős-Renyi graph generator with probability p. The average number of paths of length l from node k to the sink, where $k < L$, is $\mathbb{E}[N_l^{(k)}] = \binom{L-k-1}{l-2} p^{l-1}$ and the average total number of paths from node k to the sink is $\mathbb{E}[N^{(k)}] = p(1+p)^{L-k-1}$.*

We can observe that if $p = 1$, the randomly wired network converges to the ResNet architecture. This allows to think of randomly wired architectures as generalizations of ResNets as they enable increased flexibility in the number and distribution of paths instead of enforcing the use of all 2^{L-2}. By means of the parameter p and the weights ω_{ij} of the architecture edges, we can modulate the contributions of various path lengths and the size of the output receptive field, which is obtained as a combination of the receptive fields of shallower networks induced by each of the paths of the ensemble. In addition, the trainable weights ω_{ij} can enable adaptive receptive fields, that can be tuned by the training procedure.

2.2 Sequential path

Differently from ResNets, in a randomly wired neural network with L architecture nodes the longest path may be shorter than L, leading to a smaller receptive field. In order to overcome this issue, we propose to modify the generation process of the random architecture by imposing that it should also include the sequential path, i.e., the path traversing all architecture nodes. This design of the architecture skews the initial path length distribution towards longer paths, which has the effect of promoting their usage. Nevertheless, the trainable architecture edge weights will ultimately define the importance of such contribution.

2.3 MonteCarlo DropPath regularization

The randomly wired architecture offers new degrees of freedom to introduce regularization techniques. In particular, one could delete a few architecture edges during training with probability p_{drop} as a way to avoid co-adaptation of architecture nodes. This is reminiscent of DropOut (Srivastava et al., 2014) and DropConnect (Wan et al., 2013), although it is carried out at a higher level of abstraction, i.e., connections between "layers" instead of neurons. We propose to use a MonteCarlo approach where paths are also dropped in testing. Inference is performed multiple times for different realizations of dropped architecture edges and results are averaged. This allows to sample from the full predictive distribution induced by DropPath, as in MonteCarlo DropOut (Gal & Ghahramani, 2015).

3 Experimental results

Experimental evaluation of GNNs is a topic that has recently received great attention. In particular, Vignac et al. (2020) showed that commonly used citation network datasets like CORA, CITESEER, PUBMED are too simple and skew results towards simpler architectures or even promote ignoring the underlying graph. TU datasets are also recognized to be too small (Errica et al., 2019) and the high variability across splits does not allow for sound comparisons. In order to evaluate the gains offered by randomly wired architectures, we adopt recently proposed benchmarking frameworks such as the one in (Dwivedi et al., 2020) and Open Graph Benchmarks (Weihua Hu, 2020). We focus on the ZINC (graph regression) dataset from (Dwivedi et al., 2020), and the ogbg-molpcba (graph classification) dataset from (Weihua Hu, 2020).

First, we use the ZINC dataset to analyse the performance differences between the baseline ResNet architecture, i.e., a feedforward architecture with skip connections after every layer, and the randomly wired architecture. For this experiment, we test four of the most commonly used graph convolution definitions: GCN (Kipf & Welling, 2017), GIN (Xu et al., 2019)[1], Gated GCN (Bresson & Laurent, 2017), GraphSage (Hamilton et al., 2017). Notice that we do not attempt to optimize a specific method, nor we are interested in comparing one graph convolution to another. A fair comparison is ensured by running both methods with the same number of trainable parameters and with the same hyperparameters, keeping exactly the same ones used in (Dwivedi et al., 2020). The learning rate of both methods is adaptively decayed between 10^{-3} and 10^{-5} and stopping criterion is validation loss not improving for 5 epochs after reaching the minimum learning rate. Results are averaged over 4 runs with different weight initialization and different random architecture graphs, drawn with $p = 0.6$. We use sequential path embedding but no DropPath.

[1] GIN and RAN-GIN compute the output as in (Xu et al., 2018), using all architecture nodes.

Table 1: ZINC Mean Absolute Error.

	$L = 4$	$L = 8$	$L = 16$	$L = 32$
GCN	$\mathbf{0.469}^{\pm 0.002}$	$0.465^{\pm 0.012}$	$0.445^{\pm 0.022}$	$0.426^{\pm 0.011}$
RAN-GCN	$0.509^{\pm 0.015}$	$\mathbf{0.447}^{\pm 0.019}_{1.5\sigma}$	$\mathbf{0.398}^{\pm 0.015}_{2.1\sigma}$	$\mathbf{0.385}^{\pm 0.015}_{3.7\sigma}$
GIN	$0.375^{\pm 0.014}$	$0.444^{\pm 0.017}$	$0.461^{\pm 0.022}$	$0.633^{\pm 0.089}$
RAN-GIN	$0.381^{\pm 0.021}$	$\mathbf{0.398}^{\pm 0.004}_{2.7\sigma}$	$\mathbf{0.426}^{\pm 0.020}_{1.6\sigma}$	$\mathbf{0.540}^{\pm 0.155}_{1.0\sigma}$
GatedGCN	$0.368^{\pm 0.007}$	$0.339^{\pm 0.027}$	$0.284^{\pm 0.014}$	$0.277^{\pm 0.025}$
RAN-GatedGCN	$0.364^{\pm 0.007}$	$\mathbf{0.310}^{\pm 0.010}_{1.1\sigma}$	$\mathbf{0.218}^{\pm 0.017}_{4.7\sigma}$	$\mathbf{0.215}^{\pm 0.025}_{2.5\sigma}$
GraphSage	$0.428^{\pm 0.007}$	$0.363^{\pm 0.005}$	$0.355^{\pm 0.003}$	$0.351^{\pm 0.009}$
RAN-GraphSage	$0.429^{\pm 0.010}$	$\mathbf{0.368}^{\pm 0.015}_{1.0\sigma}$	$\mathbf{0.340}^{\pm 0.009}_{5.0\sigma}$	$\mathbf{0.333}^{\pm 0.008}_{2.0\sigma}$

Table 2: Median Relative Gain over $L = 4$.

	ResNet	**Random**
$L = 8$	-0.50%	**5.43%**
$L = 16$	0.37%	**8.89%**
$L = 32$	3.24%	**11.36%**

Table 3: ogbg-molpcba Average Precision.

	Test AP	Val AP	No. params
GCN	$0.2020^{\pm 0.0024}$	$0.2059^{\pm 0.0033}$	$565,928$
GIN	$0.2266^{\pm 0.0028}$	$0.2305^{\pm 0.0027}$	$1,923,433$
ChebNet	$0.2306^{\pm 0.0016}$	$0.2372^{\pm 0.0018}$	$1,475,003$
SIGN	$0.2047^{\pm 0.0036}$	$0.2163^{\pm 0.0022}$	$5,516,228$
RAN-GIN	$0.2493^{\pm 0.0076}$	$0.2514^{\pm 0.0093}$	$1,868,774$
GCN+VN+FLAG	$0.2384^{\pm 0.0037}$	$0.2556^{\pm 0.0040}$	$2,017,028$
GIN+VN+FLAG	$0.2834^{\pm 0.0038}$	$0.2912^{\pm 0.0026}$	$3,374,533$
DeeperGCN+VN+FLAG	$0.2842^{\pm 0.0043}$	$0.2952^{\pm 0.0029}$	$5,550,208$
RAN-GIN+VN+FLAG	$0.2879^{\pm 0.0048}$	$\mathbf{0.3041}^{\pm 0.0031}$	$5,572,026$
GINE++VN	$\mathbf{0.2917}^{\pm 0.0015}$	$\mathbf{0.3065}^{\pm 0.0030}$	$6,147,029$

The results shown in Table 1 show that randomly wired GNNs have compelling performance in many regards. The superscript reports the standard deviation among runs and the subscript reports the level of significance by measuring how many baseline standard deviations the average value of the random architecture deviates from the average value of the baseline. Results are in bold if they are at least 1σ significant. First of all, randomly wired GNNs typically provide lower error than their ResNet counterparts for the same number of parameters. Moreover, they are more effective at increasing capacity than stacking layers: while they are essentially equivalent to ResNets for very short networks, they enable larger gains when additional layers are introduced. This is highlighted by Table 2, which shows the relative improvement in mean absolute error averaged over all the graph convolution definitions, with respect to the short 4-layer network, where random wiring and ResNets are almost equivalent. We can notice that deeper ResNets always provide lower gains with respect to their shallow counterpart than the randomly wired GNNs.

Finally, we compare the proposed method against other state-of-the-art techniques, including methods that address the oversmoothing problem to build deeper GNNs (DeeperGCN) (Li et al., 2020) or argue that going wide instead of deep is more effective to increase the capacity (SIGN) (Rossi et al., 2020). This experiment is done on the ogbg-molpcba dataset from Open Graph Benchmarks (Weihua Hu, 2020) and results are taken from the public leaderboard. We use a randomly wired version of GIN and compare results with two different setups: a vanilla RAN-GIN with the same number of parameters as GIN, and a larger RAN-GIN with the virtual node trick (Gilmer et al., 2017) and FLAG augmentations (Kong et al., 2020). Both versions additionally use DropPath with $p_{drop} = 0.01$. The results are reported in Table 3. We can see that RAN-GIN (12 layers) significantly outperforms GIN and a number of other techniques for a comparable number of parameters. Furthermore, RAN-GIN with virtual node and FLAG augmentations reaches state-of-the-art performance on this benchmark, outperforming DeeperGCN and being equal to the recently proposed GINE (Brossard et al., 2020)[2].

4 Conclusion

Randomly wired architectures are a promising way to increase the capacity of graph neural networks, going beyond the traditional depth concept. Premilinary experiments presented in this paper show convincing improved to a variety of graph convolutions and warrant further tests.

[2]Notice that we did not test RAN-GINE.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Rémy Brossard, Oriel Frigo, and David Dehaene. Graph convolutions that can finally model local structure. *arXiv preprint arXiv:2011.15069*, 2020.

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

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272. PMLR, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

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

Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. *arXiv preprint arXiv:2010.09891*, 2020.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs go as deep as CNNs? In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9267–9276, 2019.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in neural information processing systems*, pp. 550–558, 2016.

Clément Vignac, Guillermo Ortiz-Jiménez, and Pascal Frossard. On the choice of graph neural network architectures. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8489–8493. IEEE, 2020.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pp. 1058–1066, 2013.

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

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1284–1293, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462, 2018.

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