RAN-GNNs: breaking the capacity limits of graph neural networks

Diego Valsesia, *Member, IEEE*, Giulia Fracastoro, *Member, IEEE*, Enrico Magli, *Fellow, IEEE*

***Abstract*—Graph neural networks have become a staple in problems addressing learning and analysis of data defined over graphs. However, several results suggest an inherent difficulty 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. We also provide extensive experimental evidence of the superior performance of randomly wired architectures over multiple tasks and five graph convolution definitions, using recent benchmarking frameworks that addresses the reliability of previous testing methodologies.**

I. Introduction

Data defined over the nodes of graphs are ubiquitous. Social network profiles [1], molecular interactions [2], citation networks [3], 3D point clouds [4] are just examples of a wide variety of data types where describing the domain as a graph allows to encode constraints and patterns among the data points. Exploiting the graph structure is crucial in order to extract powerful representations of the data. However, this is not a trivial task and only recently graph neural networks (GNNs) have started showing promising approaches to the problem. GNNs [5] extend the deep learning toolbox to deal with the irregularity of the graph domain. Much of the work has been focused on defining a graph convolution operation [6], i.e., a layer that is well-defined over the graph domain but also retains some of the key properties of convolution such as weight reuse and locality. A wide variety of such graph convolution operators has been defined over the years, mostly based on neighborhood aggregation schemes where the features of a node are transformed by processing the features of its neighbors. Such schemes have been shown to be as powerful as the Weisfeiler-Lehman graph isomorphism test [7], [8], enabling them to simultaneuosly learn data features and graph topology.

However, contrary to classic literature on CNNs, few works [9]–[12] addressed GNNs architectures and their role in extracting powerful representations. Several works, starting with the early GCN [13], noticed an inability to build deep GNNs, often resulting in worse performance than that of methods that disregard the graph domain, when trying to build anything but very shallow networks. 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. [14] 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. [11] 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, i.e. the total number of parameters, of graph neural networks is still standing, since stacking many layers has been proven to provide limited improvements [14]–[19].

In this paper, we argue that the recently proposed randomly wired architectures [20], [21] 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 [11] of being able of merging receptive fields of varied size. Indeed, the randomly wired GNNs (RAN-GNNs) can be seen as an extreme generalization of their jumping network 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.

Diego Valsesia and Giulia Fracastoro contributed equally to this work. The authors are with Politecnico di Torino - Department of Electronics and Telecommunications, Italy. Email:name.surname@polito.it.

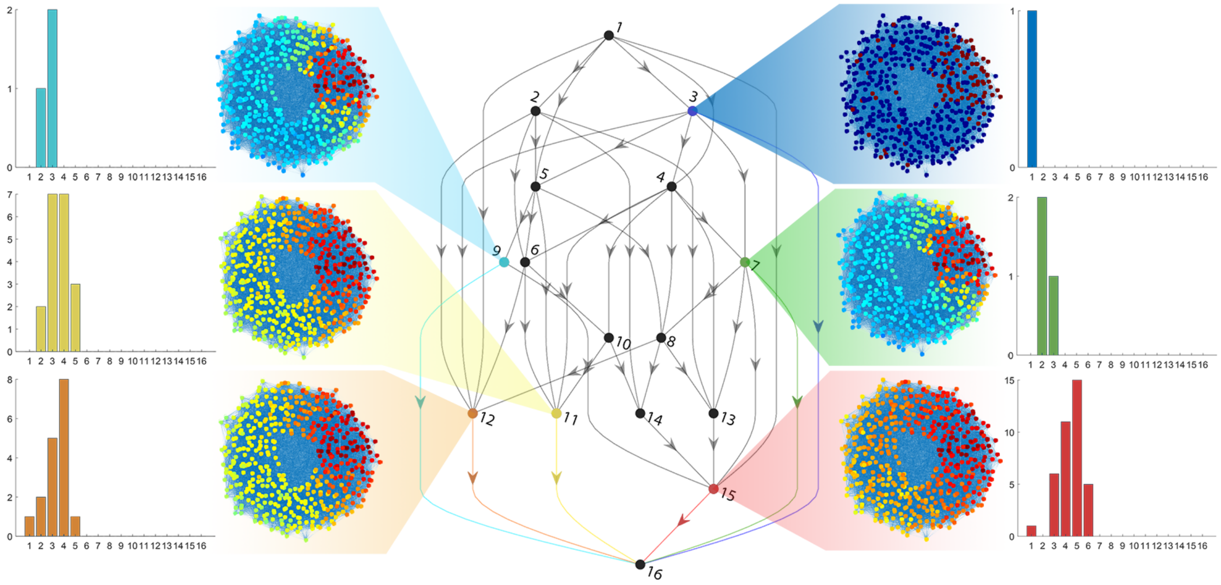

Fig. 1. Random architectures aggregate ensembles of paths. This creates a variety of receptive fields (effective neighborhood sizes on the domain graph) that are combined to compute the output. Figure shows the domain graph where nodes are colored (red means high weight, blue low weight) according to 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.

Our novel contributions can be summarized as follows: i) we are the first to analyze randomly wired architectures and show that they are generalizations of ResNets when looked at as ensembles of paths [22]; ii) we show that path ensembling allows to merge receptive fields of varied size and that it can do so *adaptively*, i.e., trainable weights on the architecture edges can tune the desired size of the receptive fields to be merged to achieve an optimal configuration for the problem; iii) we introduce improvements to the basic design of randomly wired architectures by optionally embedding a path that sequentially goes through all layers in order to promote larger receptive fields when needed, and by presenting MonteCarlo DropPath, which decorrelates path contributions by randomly dropping architecture edges; iv) we provide extensive experimental evidence, using recently introduced benchmarking frameworks [12], [23] to ensure significance and reproducibility, that randomly wired architectures consistently outperform ResNets, often by large margins, for five of the most popular graph convolution definitions on multiple tasks.

II. Background

A. Graph Neural Networks

A major shortcoming of CNNs is that they are unable to process data defined on irregular domains. In particular, one case that is drawing attention is when the data structure can be described by a graph and the data are defined as vectors on the graph nodes. This setting can be found in many applications, including 3D point clouds [24], [25], computational biology [2], [26], and social networks [13]. However, extending CNNs from data with a regular structure, such as images and video, to graph-structured data is not straightforward if one wants to preserve useful properties such as locality and weight reuse.

GNNs redefine the convolution operation so that the new layer definition can be used on domains described by graphs. The most widely adopted graph convolutions in the literature rely on message passing, where a weighted aggregation of the feature vectors in a neighborhood is computed. The GCN [13] is arguably the simplest definition, applying the same linear transformation to all the node features, followed by neighborhood aggregation and non-linear activation:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}\mathbf{W}\mathbf{h}_j^{(l)}\right).$$

Variants of this definition have been developed, e.g., GraphSage [1] introduces a simple form of skip connection by concatenating the feature vector of node i to the feature vectors of its neighbors; GIN [8] uses a multilayer perceptron instead of a linear transform, replaces average with sum to ensure injectivity and proposes a different way of computing the output by using all the feature vectors produced by the intermediate layers. These definitions are all isotropic because they treat every edge in the same way. It has been observed that better representation capacity can be achieved using anistropic definitions, where every edge can have a different transformation, at the cost of increased computational complexity. The Gated GCN [27] and GAT [28] definitions fall in this category.

B. Randomly wired architectures

In recent work, Xie et al. [20] explore whether it is possible to avoid handcrafted design of neural network architectures and, at the same time, avoid expensive neural architecture search methods [29], by designing random architecture generators. They show that “layers” performing convolution, normalization and non-linear activation can be connected in a

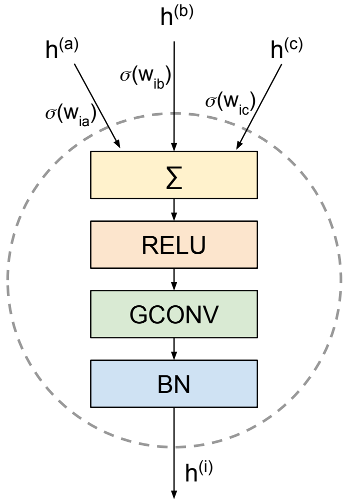

Fig. 2. An architecture node is equivalent to a GNN layer.

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.

III. Randomly wired GNNs

In this section, we first introduce randomly wired graph neural networks (RAN-GNNs) and the notation we are going to use. We then analyze their behavior when viewed as ensembles of paths.

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 [20]:

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

being σ a sigmoid function, $\mathcal{A}_i$ 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.

The architecture DAG is generated using a random graph generator. In this paper, we will focus on the Erdős-Renyi model where the adjacency matrix of the DAG is a strictly upper triangular matrix with entries being realizations of a Bernoulli random variable with probability p. 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. Other random generators may be used, e.g., small-world and scale-free random networks have been studied in [20]. However, a different generator will display a different behavior concerning the properties we study in Sec. III-A.

A. 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 [22]. More specifically, in a ResNet with L 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 $\frac{L}{2}+1$ [22]. In this section, we show that 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 proposition.

Proposition III.1. *Let us consider a randomly wired network with L architecture nodes, where the architecture DAG is generated according to an 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 $\mathbb{E}[N^{(k)}] = p(1+p)^{L-k-1}$.*

Proof. Let us first consider the number of paths of length l from node k to the sink. We define the path length as the number of nodes in the path. In a randomly wired network with L architecture nodes, we have that the first node of all the paths is node k and the last one is node L (i.e., the sink node). Therefore, the minimum path length is 2. If $l \geq 2$, the number of all possible paths of length l between node k and the sink is $\binom{L-k-1}{l-2}$. Since in a path of length l there are $l-1$ edges and each edge has probability p of being generated by the Erdős-Renyi model, each one of the paths of length l has probability p^{l-1} of being present in the network. Thus, the expected number of paths with length l between node k and the sink is $\mathbb{E}[N_l^{(k)}] = \binom{L-k-1}{l-2} p^{l-1}$. If we set $k=1$, we obtain the average number of paths of length l from source to sink $\mathbb{E}[N_l] = \binom{L-2}{l-2} p^{l-1}$. We can now compute the average total number of paths $\mathbb{E}[N^{(k)}]$ as follows

$$\mathbb{E}[N^{(k)}] = \sum_{l=2}^{L-k+1} \binom{L-k-1}{l-2} p^{l-1} = \sum_{\tilde{l}=0}^{\tilde{n}} \binom{\tilde{n}}{\tilde{l}} p^{\tilde{l}+1}$$
$$= p \sum_{\tilde{l}=0}^{\tilde{n}} \binom{\tilde{n}}{\tilde{l}} p^{\tilde{l}} = p(1+p)^{\tilde{n}} = p(1+p)^{L-k-1},$$

where $\tilde{n} = L-k-1$, $\tilde{l} = l-2$ and the fourth equality follows from the binomial theorem. If we set $k=1$, we obtain the average total number of paths from source to sink $\mathbb{E}[N_p] = p(1+p)^{L-2}$. □

We can observe that if $p = 1$, the distribution of paths of a randomly wired network converges to the one of 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} paths.

B. Receptive field analysis

In the case of GNNs, we define the receptive field of a domain node as the neighborhood that affects the output features of that node. As discussed in Sec. I, the work in [11] highlights that one of the possible causes of the depth problem in GNNs is that the size of the receptive field is not adaptive and may rapidly become excessively large. Inspired by this observation, in this section we analyze the receptive field of a randomly wired graph neural network. We show that the receptive field of the output is a combination of the receptive fields of shallower networks, induced by each of the paths. This allows to effectively merge the contributions from receptive fields of varied size. Moreover, we show that the trainable parameters along the path edges modulate the contributions of various path lengths and enable adaptive receptive fields, that can be tuned by the training procedure.

We first introduce a definition of the receptive field of a feedforward graph neural network[1].

Definition III.1. *Given a feedforward graph neural network with L layers, the receptive field of radius L of a domain node is its L-hop neighborhood.*

In a randomly wired architecture, each path induces a corresponding receptive field whose radius depends on the length of the path. Then, the receptive field at the output of the network is obtained by combining the receptive fields of all the paths. In order to analyze the contribution of paths of different lengths to the receptive field of the network, we introduce the concept of distribution of the receptive field radius of the paths. Notice that if we consider a feedforward network with L layers, the distribution of the receptive field radius is a delta centered in L.

The following proposition allows to analyze the distribution of the receptive field radius in a randomly wired architecture.

Proposition III.2. *The derivative $\frac{\partial y}{\partial x_0}$ of the output y of a randomly wired architecture with respect to the input x_0 is*

$$\frac{\partial y}{\partial x_0} = \sum_{p \in \mathcal{P}} \frac{\partial y_p}{\partial x_0} = \sum_{p \in \mathcal{P}} \prod_{\{i,j\} \in \mathcal{E}^p} \omega_{ij} \frac{\partial \bar{y}_p}{\partial x_0} = \sum_{l=2}^{L} \sum_{p \in \mathcal{P}^l} \lambda_p \frac{\partial \bar{y}_p}{\partial x_0}, \tag{2}$$

where y_p is the output of path p, $\bar{y}_p$ is the output of path p when we consider all the aggregation weights equal to 1, $\lambda_p = \frac{\partial y_p}{\partial x_0} / \frac{\partial \bar{y}_p}{\partial x_0}$, $\mathcal{P}$ is the set of all paths from source to sink, L is the number of architecture nodes, $\mathcal{P}^l$ is the set of paths from source to sink of length l and $\mathcal{E}^p$ is the set of edges of the path p.

Proof. Direct computation. □

From Proposition III.2, we can observe that the contribution of each path to the gradient is weighted by its corresponding architecture edge weights. Thus, we can define the following distribution ρ of the receptive field radius:

$$\rho_l = \sum_{p \in \mathcal{P}^l} \lambda_p = \sum_{p \in \mathcal{P}^l} \prod_{\{i,j\} \in \mathcal{E}^p} \omega_{ij} \qquad \text{for} \quad l = 2, ..., n, \tag{3}$$

where we have assumed that the gradient $\frac{\partial \bar{y}_p}{\partial x_0}$ depends only on the path length, as done in [22]. This is a reasonable assumption if all the architecture nodes perform the same operation. The distribution of the receptive field radius is therefore influenced by the architecture edge weights. Figure 3 shows an example of how such weights can modify the radius distribution. If we consider $\omega_{ij} = 1$ for all i and j, we obtain that the radius distribution is equal to the path length distribution. In order to provide some insight into the role of parameter p in the distribution of the receptive field radius, we focus on this special case and analyze the distribution of the path lengths in a randomly wired architecture by introducing the following Proposition.

Proposition III.3. *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 length of the paths from node k to the sink is $\mathbb{E}[l^{(k)}] \approx \frac{p}{1+p}(L-k-1)+2$.*

Proof. From Proposition 3.1, we can compute the average length of the paths from node k to the sink as follows

$$\begin{aligned} \mathbb{E}[l^{(k)}] &= \sum_{l=2}^{L-k+1} l \mathbb{E}\left[\frac{N_l^k}{N^{(k)}}\right] \approx \frac{\sum_{l=2}^{L-k+1} l \mathbb{E}[N_l^{(k)}]}{\mathbb{E}[N^{(k)}]} \\ &= \frac{\sum_{l=2}^{L-k+1} \binom{L-k-1}{l-2} p^{l-1} l}{p(1+p)^{L-k-1}}, \end{aligned} \tag{4}$$

where we have neglected the higher order terms [30]. The numerator in (4) can be computed as follows

$$\begin{aligned} \sum_{l=2}^{L-k+1} \binom{L-k-1}{l-2} p^{l-1} l &= \sum_{\tilde{l}=0}^{\tilde{n}} \binom{\tilde{n}}{\tilde{l}} p^{\tilde{l}+1} (\tilde{l}+2) \\ &= \sum_{\tilde{l}=0}^{\tilde{n}} \binom{\tilde{n}}{\tilde{l}} p^{\tilde{l}+1} \tilde{l} + 2 \sum_{\tilde{l}=0}^{\tilde{n}} \binom{\tilde{n}}{\tilde{l}} p^{\tilde{l}+1} \\ &= p^2 \sum_{\tilde{l}=0}^{\tilde{n}} \binom{\tilde{n}}{\tilde{l}} p^{\tilde{l}-1} \tilde{l} + 2p \sum_{\tilde{l}=0}^{\tilde{n}} \binom{\tilde{n}}{\tilde{l}} p^{\tilde{l}} \\ &= p^2 \tilde{n} (1+p)^{\tilde{n}-1} + 2p(1+p)^{\tilde{n}} \\ &= p^2 (L-k-1)(1+p)^{L-k-2} \\ &\quad + 2p(1+p)^{L-k-1}, \end{aligned}$$

where $\tilde{n} = L-k-1$, $\tilde{l} = l-2$ and the fourth equality is obtained differentiating the binomial theorem with respect to p . Then, we obtain

$$\mathbb{E}[l^{(k)}] = \frac{p}{1+p}(L-k-1)+2.$$

[1] We use the term "feedforward neural network" to indicate an architecture made of a simple path graph, without skip connections: this is a representation of one path.

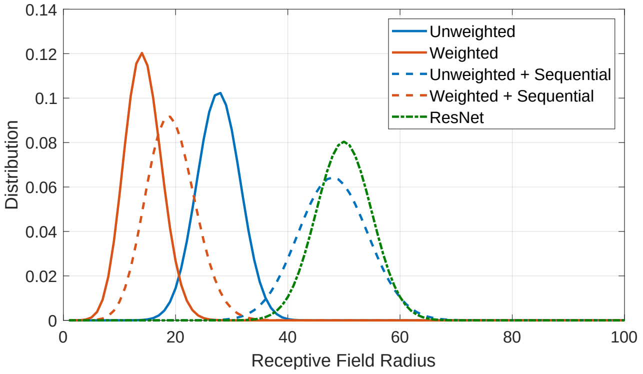

Fig. 3. Distribution of receptive field radius ($p = 0.4$, $\omega_{ij} = 1$ for unweighted, $\omega_{ij} = 0.5$ for weighted).

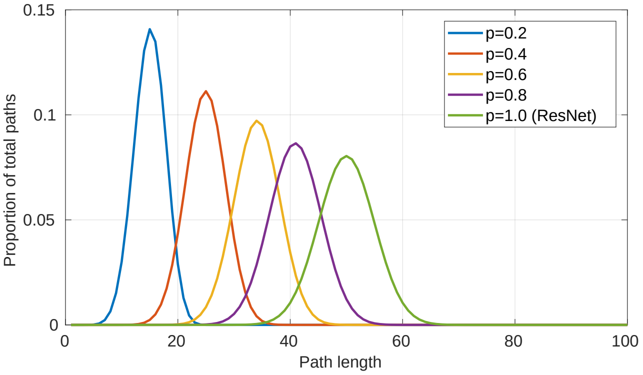

Fig. 4. Path distribution as function of architecture edge probability.

If we consider $k = 1$, i.e. the sink, we obtain $\mathbb{E}[l] = \frac{p}{1+p}(L - 2) + 2$. □

Therefore, if $p = 1$ and $\omega_{ij} = 1$ for all i and j the radius distribution is a Binomial distribution centered in $\frac{L}{2} + 1$ (as in ResNets), instead when $p < 1$ the mean of the distribution is lower. The path length distribution for different p values is shown in Fig. 4. This shows that, differently from feedforward networks, the receptive field of ResNets and randomly wired architectures is a combination of receptive fields of varied sizes, where most of the contribution is given by shallow paths, i.e. smaller receptive fields. The parameter p of the randomly wired neural network influences the distribution of the receptive field radius: a lower p value skews the distribution towards shallower paths, instead a higher p value skews the distribution towards longer paths.

After having considered the special case where $\omega_{ij} = 1$ for all i and j, we now focus on the general case. Since the edge architecture weights are trainable parameters, they can be adapted to optimize the distribution of the receptive field radius. This is one of the strongest advantages provided by randomly wired architectures with respect to ResNets. This is particularly relevant in the context of GNNs, where we may have a non-uniform growth of the receptive field caused by the irregularity of the graph structure [11]. Notice that the randomly wired architecture can be seen as a generalization of the jumping knowledge networks proposed in [11], where all the architecture nodes, not only the last one, merge contributions from previous nodes. We also remark that, even if we modify the ResNet architecture by adding trainable weights to each branch of the residual module, we cannot retrieve the behaviour of the randomly wired architecture. In fact, the latter has intrinsically more granularity than a ResNet: the expected number of architecture edge weights of a randomly wired network is $\frac{pL(L+1)}{2}$, instead a weighted ResNet has only $2(L-2)$ weights. Ideally, we would like to weigh each path independently (i.e., directly optimizing the value of λ_p in Eq. (2)). However, this is unfeasible because the number of parameters would become excessively high and the randomly wired architecture provides an effective tradeoff. Given an architecture node, weighting in a different way each input edge is important because to each edge corresponds a different length distribution of the paths going through such edge, as shown by the following Proposition.

Proposition III.4. *Let us consider a randomly wired network with n architecture nodes, where the architecture DAG is generated according to a Erdős-Renyi graph generator with probability p. Given an edge $\{i, j\}$ between the architecture nodes i and j where $i < j$, the average length of the paths from the source to the sink going through that edge is $\mathbb{E}[l_{ij}] \approx \frac{p}{1+p}(L - (j - i) - 3) + 4$.*

Proof. From Proposition 3.3, we can compute the average length of the paths going through the edge $\{i, j\}$ as follows

$$\mathbb{E}[l_{ij}] = \mathbb{E}[l^{n-i+1} + l^j] \approx \frac{p}{1+p}(n - (j - i) - 3) + 4.$$

□

C. Sequential path

In the previous sections we have shown that a randomly wired architecture behaves like an ensemble of paths merging contribution from receptive fields of varied size, where most of the contribution is provided by shallow paths. As discussed previously, this provides numerous advantages with respect to feedforward networks and ResNets. However, some graph-based tasks may actually benefit from a larger receptive field [14], so it is interesting to provide randomly wired architectures with mechanisms to directly promote longer paths. 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. Fig. 3 shows an example of how including the sequential path changes the distribution of the receptive field radius.

D. 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 [31] and DropConnect [32], although it is carried out at a higher level of abstraction, i.e., connections between "layers" instead of neurons. It is also reminiscent of techniques used in Neural Architecture Search [33] and the approach used in ImageNet experiments in [20], although implementation details are unclear for the latter.

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 [34]. The following proposition shows that MonteCarlo DropPath decorrelates the contributions of paths in Eq. (2) even if they share architecture edges, thus allowing finer control over the modulation of the receptive field radius.

Proposition III.5. *Let us consider two distinct paths p_1 and p_2 of a randomly wired network where the edges of the paths can be deleted with probability p_{drop}. Then, even if the two paths share some architecture edges, their contributions to the derivative $\frac{\partial y}{\partial x_0}$, as defined in Proposition III.2, are decorrelated.*

Proof. Let us consider two paths p_1 and p_2 with at least one common edge, we can compute the covariance between these two paths as follows:

$$\mathrm{Cov}(\lambda_{p_1}, \lambda_{p_2}) = \mathbb{E}[\lambda_{p_1}\lambda_{p_2}] - \mathbb{E}[\lambda_{p_1}]\mathbb{E}[\lambda_{p_2}]$$

$$= \prod_{\{i,j\}\in\mathcal{E}^{p_1}} \omega_{ij} \prod_{\{i,j\}\in\mathcal{E}^{p_2}} \omega_{ij} \mathbb{E}\left[\prod_{\{i,j\}\in\mathcal{I}(p_1,p_2)} z_{ij}^2 \prod_{\{i,j\}\in\mathcal{D}(p_1,p_2)} z_{ij} \right]$$

$$- \prod_{\{i,j\}\in\mathcal{E}^{p_1}} \omega_{ij} \prod_{\{i,j\}\in\mathcal{E}^{p_2}} \omega_{ij} \mathbb{E}\left[\prod_{\{i,j\}\in\mathcal{E}^{p_1}} z_{ij} \right] \mathbb{E}\left[\prod_{\{i,j\}\in\mathcal{E}^{p_2}} z_{ij} \right] = 0,$$

where $\mathcal{I}(p_1, p_2) = \mathcal{E}^{p_1} \cap \mathcal{E}^{p_2}$, $\mathcal{D}(p_1, p_2) = (\mathcal{E}^{p_1} \cup \mathcal{E}^{p_2}) - (\mathcal{E}^{p_1} \cap \mathcal{E}^{p_2})$, $\lambda_{p_1} = \prod_{\{i,j\}\in p_1} z_{ij}\omega_{ij}$, $\lambda_{p_2} = \prod_{\{i,j\}\in p_2} z_{ij}\omega_{ij}$, $z_{ij} \sim \text{Bernoulli}(1 - p_{\text{drop}})$, and we have assumed all ω_{ij} deterministic. □

IV. Experimental results

Experimental evaluation of GNNs is a topic that has recently received great attention. The emerging consensus is that benchmarking methods routinely used in past literature are inadequate and lack reproducibility. In particular, [35] 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 [36] and the high variability across splits does not allow for sound comparisons across methods. In order to evaluate the gains offered by randomly wired architectures across, we adopt recently proposed benchmarking frameworks such as the one in [12] and Open Graph Benchmarks [23].

First, we use three datasets in [12] 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. We omit results on architectures without skip connections as these have already been shown to have poorer performance [12]. We focus on the ZINC, CIFAR10 and COLLAB datasets. ZINC is one of the most popular real-world molecular datasets, and considers the task of property regression (constrained solubility) for the molecules represented as graphs. CIFAR10 is a well-known dataset for image classification, and, in this context, images are described by graphs of superpixels. COLLAB represents collaborations between authors and casts prediction of future collaborations as a link prediction problem. For this experiment, we test five of the most commonly used graph convolution definitions: GCN [13], GIN [8][2], Gated GCN [27], GraphSage [1], and GAT [28], thus analyzing both isotropic and anisotropic designs. 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 [12], except for GAT in the COLLAB experiments which has the features reduced to 80 due to memory constraints. The learning rate of both methods is adaptively decayed between 10^{-3} and 10^{-5} and the 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 $p = 0.6$. The random architectures use sequential paths (Sec. III-B), but no DropPath (Sec. III-D) for the ZINC experiment, and DropPath but no sequential paths for CIFAR10 and COLLAB. When DropPath is used, we set $p_{\text{drop}} = 0.01$ and, in testing, the results are averaged over 16 different realizations of dropped architecture edges.

The results in Tables I, II and III show the performance achieved by randomly wired architectures and their ResNets counterparts for increasing model capacity (number of architecture nodes or layers L). We remark that COLLAB is tested only up to $L = 16$ due to memory constraints. We can notice 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 or higher accuracy 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 (e.g., for $L = 4$), they enable larger gains when additional layers are introduced. This is highlighted by Table IV, which shows the relative improvement in mean absolute error or accuracy averaged over all the graph convolution definitions, with respect to the short 4-layer network, where random wiring and ResNets are almost equivalent. This table

[2] GIN and RAN-GIN compute the output as in [11], using all architecture nodes.

TABLE I
ZINC MEAN ABSOLUTE ERROR.

	$L = 4$	$L = 8$	$L = 16$	$L = 32$
GCN	$\mathbf{0.469}^{\pm 0.002}_{2.9\sigma}$	$0.465^{\pm 0.012}$	$0.445^{\pm 0.022}$	$0.426^{\pm 0.011}$
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	$0.375^{\pm 0.014}_{0.4\sigma}$	$0.444^{\pm 0.017}$	$0.461^{\pm 0.022}$	$0.633^{\pm 0.089}$
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	$0.364^{\pm 0.007}_{0.5\sigma}$	$\mathbf{0.310}^{\pm 0.010}_{1.1\sigma}$	$\mathbf{0.218}^{\pm 0.017}_{4.7\sigma}$	$\mathbf{0.215}^{\pm 0.025}_{2.5\sigma}$
GraphSage	$0.428^{\pm 0.007}_{0.1\sigma}$	$0.363^{\pm 0.005}$	$0.355^{\pm 0.003}$	$0.351^{\pm 0.009}$
RAN-GraphSage	$0.429^{\pm 0.010}$	$\mathbf{0.368}^{\pm 0.015}_{1.0\sigma}$	$\mathbf{0.340}^{\pm 0.009}_{5.0\sigma}$	$\mathbf{0.333}^{\pm 0.008}_{2.0\sigma}$
GAT	$0.482^{\pm 0.007}_{0.6\sigma}$	$\mathbf{0.420}^{\pm 0.009}_{2.3\sigma}$	$0.394^{\pm 0.011}_{0.9\sigma}$	$0.391^{\pm 0.006}_{0.7\sigma}$
RAN-GAT	$0.487^{\pm 0.009}$	$0.441^{\pm 0.005}$	$0.405^{\pm 0.012}$	$0.396^{\pm 0.007}$

TABLE II
CIFAR10 ACCURACY.

	$L = 4$	$L = 8$	$L = 16$	$L = 32$
GCN	$54.28^{\pm 0.35}$	$54.85^{\pm 0.20}$	$54.74^{\pm 0.52}$	$54.76^{\pm 0.53}$
RAN-GCN	$\mathbf{55.31}^{\pm 0.25}_{2.9\sigma}$	$\mathbf{57.81}^{\pm 0.08}_{14.8\sigma}$	$\mathbf{57.29}^{\pm 0.44}_{4.9\sigma}$	$\mathbf{58.49}^{\pm 0.21}_{7.0\sigma}$
GIN	$\mathbf{70.66}^{\pm 0.78}_{2.94\sigma}$	$66.67^{\pm 0.73}$	$63.99^{\pm 1.45}$	$58.18^{\pm 2.92}$
RAN-GIN	$67.48^{\pm 1.08}$	$\mathbf{67.36}^{\pm 0.70}_{1.0\sigma}$	$\mathbf{67.25}^{\pm 0.74}_{2.2\sigma}$	$\mathbf{62.73}^{\pm 1.57}_{1.6\sigma}$
GatedGCN	$\mathbf{69.26}^{\pm 0.36}_{2.0\sigma}$	$68.27^{\pm 0.80}$	$69.16^{\pm 0.66}$	$69.46^{\pm 0.47}$
RAN-GatedGCN	$68.55^{\pm 0.03}$	$68.86^{\pm 1.64}_{0.7\sigma}$	$\mathbf{72.00}^{\pm 0.44}_{4.3\sigma}$	$\mathbf{73.50}^{\pm 0.68}_{8.6\sigma}$
GraphSage	$\mathbf{66.14}^{\pm 0.21}_{2.4\sigma}$	$65.58^{\pm 0.46}_{0.6\sigma}$	$66.12^{\pm 0.11}_{0.0\sigma}$	$65.33^{\pm 0.34}$
RAN-GraphSage	$65.02^{\pm 0.47}$	$65.31^{\pm 0.38}$	$66.10^{\pm 1.11}$	$\mathbf{67.68}^{\pm 0.37}_{6.9\sigma}$
GAT	$64.58^{\pm 0.41}$	$63.37^{\pm 0.82}$	$63.82^{\pm 0.51}$	$64.87^{\pm 0.44}$
RAN-GAT	$64.68^{\pm 0.47}_{0.2\sigma}$	$\mathbf{64.44}^{\pm 0.76}_{1.3\sigma}$	$\mathbf{64.87}^{\pm 0.33}_{2.1\sigma}$	$\mathbf{66.75}^{\pm 0.41}_{4.3\sigma}$

TABLE III
COLLAB HITS@50.

	$L = 4$	$L = 8$	$L = 16$
GCN	$49.64^{\pm 0.86}$	$44.17^{\pm 0.67}$	$42.79^{\pm 1.84}$
RAN-GCN	$\mathbf{51.16}^{\pm 1.02}_{1.5\sigma}$	$\mathbf{51.62}^{\pm 0.59}_{12.6\sigma}$	$\mathbf{53.02}^{\pm 1.78}_{5.9\sigma}$
GIN	$\mathbf{56.76}^{\pm 0.97}_{1.3\sigma}$	$51.38^{\pm 1.53}$	$51.46^{\pm 1.88}$
RAN-GIN	$55.13^{\pm 1.22}$	$\mathbf{55.56}^{\pm 0.94}_{2.2\sigma}$	$52.87^{\pm 1.23}_{0.8\sigma}$
GatedGCN	$52.53^{\pm 0.47}_{0.0\sigma}$	$54.12^{\pm 2.67}$	$49.82^{\pm 2.03}$
RAN-GatedGCN	$52.52^{\pm 0.59}$	$54.25^{\pm 1.21}_{0.1\sigma}$	$51.05^{\pm 1.89}_{0.6\sigma}$
GraphSage	$57.63^{\pm 0.33}$	$56.31^{\pm 0.91}$	$55.49^{\pm 2.10}$
RAN-GraphSage	$\mathbf{58.18}^{\pm 0.49}_{1.1\sigma}$	$\mathbf{59.72}^{\pm 1.12}_{3.04\sigma}$	$\mathbf{59.95}^{\pm 1.23}_{2.12\sigma}$
GAT	$48.83^{\pm 2.03}$	$48.13^{\pm 2.80}$	$52.38^{\pm 1.22}$
RAN-GAT	$48.50^{\pm 3.28}_{0.1\sigma}$	$49.71^{\pm 1.12}_{0.6\sigma}$	$\mathbf{53.80}^{\pm 0.88}_{1.16\sigma}$

TABLE IV
MEDIAN RELATIVE GAIN OVER $L = 4$.

		$L = 8$	$L = 16$	$L = 32$
ZINC	ResNet	$+7.88\%$	$+17.06\%$	$+17.99\%$
	Random	**+12.18%**	**+20.75%**	**+22.38%**
CIFAR10	ResNet	-1.43%	-0.14%	$+0.29\%$
	Random	**+0.45%**	**+1.66%**	**+4.09%**
COLLAB	ResNet	-2.29%	-5.16%	-
	Random	**+2.49%**	**+3.04%**	-

highlights that deeper ResNets always provide smaller gains with respect to their shallow counterpart than the randomly wired GNNs. This allows us to conclude that randomly wired GNNs are a more effective way of increasing model capacity. It is also interesting to analyze the behavior of isotropic graph convolutions (GCN, GIN, GraphSage) against anisotropic one (GatedGCN, GAT). Aggregating the numbers from the three experiments, we measured a median relative gain at $L = 16$ over $L = 4$ that jumps from -3.71% of ResNet to $+3.04\%$ of Random for isotropic convolutions and from 3.56% of ResNet to $+7.98\%$ of Random for anisotropic convolutions. This shows that anisotropic definitions typically suffer less from difficulties at increasing capacity but that the randomly wired architectures still provide gains.

Moreover, we compare the proposed method against other state-of-the-art techniques to build graph neural networks, including methods that address the oversmoothing problem to build deeper GNNs (DeeperGCN) [37] or argue that going wide instead of deep is more effective to increase the capacity (SIGN) [38]. This experiment is done on the ogbg-molpcba dataset from Open Graph Benchmarks [23] 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 using the virtual node trick [39] and FLAG augmentations [40]. Both versions additionally use DropPath with $p_{drop} = 0.01$. The results are reported in Table V. 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 the validation set, while it outperforms DeeperGCN and is very close to the recently proposed GINE on the testing set [41][3]. Finally, Table VI and VII compare the proposed method against the most relevant state-of-the-art techniques on the ZINC and CIFAR10 dataset. These results show that the randomly-wired architecture can usually improve the corresponding baseline and in some cases, such as RAN-GatedGCN on CIFAR-10, it outperforms the existing methods, reaching the top of the leaderboard.

V. ABLATION EXPERIMENTS

In this section, we explore how some of the design choices for randomly wired GNNs can affect model performance.

A. Architecture Edge probability

We first investigate the impact of the probability p of drawing an edge in the random architecture. Table VIII shows the results for a basic random architecture without DropPath nor embedded sequential path. It appears that an optimal value of p exists that maximizes performance. This could be explained by a tradeoff between size of receptive field and the ability to modulate it.

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

TABLE V
OGBG-MOLPCBA LEADERBOARD.

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

TABLE VI
ZINC LEADERBOARD.

	Test MAE	No. params
PNA	$0.142^{\pm 0.010}$	$387,155$
MPNN (sum)	$0.145^{\pm 0.007}$	$480,805$
RAN-GatedGCN-PE	$0.206^{\pm 0.011}$	$505,135$
GatedGCN-PE	$0.214^{\pm 0.006}$	$505,011$
MoNet	$0.292^{\pm 0.006}$	$504,013$
3WLGNN-E	$0.303^{\pm 0.068}$	$507,603$
RAN-GraphSage	$0.340^{\pm 0.009}$	$388,855$
RingGNN-E	$0.353^{\pm 0.019}$	$527,283$
GraphSage	$0.355^{\pm 0.003}$	$388,919$

B. DropPath

The impact of DropPath on CIFAR10 is shown in Table IX. We found the improvement due to DropPath to be increasingly significant for a higher number of architecture nodes, as expected due to the increased number of edges. The value of the drop probability $p_{\text{drop}} = 0.01$ was not extensively cross-validated. However, Table X shows that higher drop rates typically lowered performance.

We also tested the impact of the number of iterations performed during testing with different architecture edges. Fig. 5 shows that there exists a saturation point beyond which performance does not improve. It also shows that MonteCarlo testing improves upon testing without any random drop of architecture edges.

C. Embedded sequential path

The impact of embedding a sequential path as explained in Sec. III-B is shown in Table XI. It can be observed that its effect of promoting receptive fields with larger radius is useful on this task for any number of architecture nodes. We remark that, while we do not report results for the sake of brevity, this is not always the case and some tasks (e.g., CIFAR10) do not benefit from promoting larger receptive fields.

TABLE VII
CIFAR-10 LEADERBOARD.

	Test Acc.	No. params
RAN-GatedGCN	$73.50^{\pm 0.68}$	$808,555$
PNA	$73.04^{\pm 0.35}$	$720,982$
MPNN (max)	$71.83^{\pm 0.37}$	$712,447$
GatedGCN	$69.46^{\pm 0.47}$	$807,997$
RAN-GraphSage	$67.68^{\pm 0.37}$	$765,441$
RAN-GAT	$66.75^{\pm 0.41}$	$774,640$
GraphSage	$65.33^{\pm 0.34}$	$765,441$
GAT	$64.87^{\pm 0.37}$	$774,640$

TABLE VIII
EDGE PROBABILITY, $L = 16$, RAN-GCN.

	$p = 0.2$	$p = 0.4$	$p = 0.6$	$p = 0.8$
ZINC	$0.440^{\pm 0.025}$	$0.427^{\pm 0.025}$	$\mathbf{0.409}^{\pm 0.010}$	$0.415^{\pm 0.012}$
CIFAR10	$56.53^{\pm 0.61}$	$56.21^{\pm 0.48}$	$\mathbf{57.44}^{\pm 0.46}$	$56.06^{\pm 0.48}$

VI. CONCLUSION

We showed how randomly wired architectures can boost the performance of GNNs by merging receptive fields of multiple size. Consistent and statistically significant improvements over a wide range of tasks and graph convolutions highlight how such constructions are more effective at increasing model capacity than building deep GNN by stacking several layers in a linear fashion, even when residual connections are used.

REFERENCES

[1] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[2] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[3] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[4] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 29–38.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[6] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[7] B. Weisfeiler and A. Lehman, "A reduction of a graph to a canonical form and an algebra arising during this reduction." *Nauchno–Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.

[8] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations (ICLR)*, 2019.

[9] G. Li, M. Müller, G. Qian, I. C. Delgadillo, A. Abualshour, A. Thabet, and B. Ghanem, "Deepgcns: Making GCNs go as deep as CNNs," *arXiv preprint arXiv:1910.06849*, 2019.

[10] N. Dehmamy, A.-L. Barabási, and R. Yu, "Understanding the representation power of graph neural networks in learning graph topology," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 387–15 397.

TABLE IX
DropPath on CIFAR10, RAN-GatedGCN. No sequential path embedding.

	$L=8$	$L=16$	$L=32$
None	$68.07^{\pm 0.94}$	$70.78^{\pm 0.38}$	$72.75^{\pm 0.37}$
DropPath	$\mathbf{68.86}^{\pm 1.64}$	$\mathbf{72.00}^{\pm 0.44}$	$\mathbf{73.50}^{\pm 0.68}$

TABLE X
DropPath on CIFAR10, RAN-GatedGCN. No sequential path embedding.

		p_{drop}		
0	0.005	0.01	0.02	0.03
$70.78^{\pm 0.38}$	$70.90^{\pm 0.46}$	$\mathbf{72.00}^{\pm 0.44}$	$71.55^{\pm 0.83}$	$71.09^{\pm 1.79}$

TABLE XI
Sequential path embedding on ZINC, RAN-GatedGCN. No DropPath.

	$L=8$	$L=16$	$L=32$
Fully random	$0.332^{\pm 0.027}$	$0.264^{\pm 0.029}$	$0.234^{\pm 0.030}$
Random+Sequential	$\mathbf{0.310}^{\pm 0.010}$	$\mathbf{0.218}^{\pm 0.017}$	$\mathbf{0.215}^{\pm 0.025}$

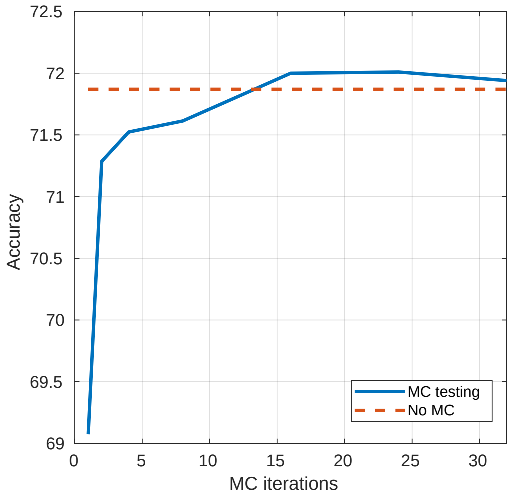

Fig. 5. Performance of MonteCarlo (MC) testing for DropPath (CIFAR10, RAN-GatedGCN, no sequential path embedding, $p_{\text{drop}} = 0.01$).

[11] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, “Representation learning on graphs with jumping knowledge networks,” in *International Conference on Machine Learning*, 2018, pp. 5453–5462.

[12] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, “Benchmarking graph neural networks,” *arXiv preprint arXiv:2003.00982*, 2020.

[13] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *International Conference on Learning Representations (ICLR)*, 2017.

[14] G. Li, M. Muller, A. Thabet, and B. Ghanem, “DeepGCNs: Can GCNs go as deep as CNNs?” in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9267–9276.

[15] Y. Rong, W. Huang, T. Xu, and J. Huang, “Dropedge: Towards deep graph convolutional networks on node classification,” in *International Conference on Learning Representations*, 2019.

[16] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, “Simple and deep graph convolutional networks,” in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, 13–18 Jul 2020, pp. 1725–1735.

[17] K. Oono and T. Suzuki, “Graph neural networks exponentially lose expressive power for node classification,” *arXiv preprint arXiv:1905.10947*, 2019.

[18] U. Alon and E. Yahav, “On the bottleneck of graph neural networks and its practical implications,” *arXiv preprint arXiv:2006.05205*, 2020.

[19] H. NT and T. Maehara, “Revisiting graph neural networks: All we have is low-pass filters,” *arXiv preprint arXiv:1905.09550*, 2019.

[20] S. Xie, A. Kirillov, R. Girshick, and K. He, “Exploring randomly wired neural networks for image recognition,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1284–1293.

[21] J. You, J. Leskovec, K. He, and S. Xie, “Graph structure of neural networks,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 10 881–10 891.

[22] A. Veit, M. J. Wilber, and S. Belongie, “Residual networks behave like ensembles of relatively shallow networks,” in *Advances in neural information processing systems*, 2016, pp. 550–558.

[23] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, “Open graph benchmark: Datasets for machine learning on graphs,” *arXiv preprint arXiv:2005.00687*, 2020.

[24] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic graph cnn for learning on point clouds,” *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5, p. 146, 2019.

[25] D. Valsesia, G. Fracastoro, and E. Magli, “Learning Localized Generative Models for 3D Point Clouds via Graph Convolution,” in *International Conference on Learning Representations (ICLR) 2019*, 2019.

[26] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, “Predicting the sequence specificities of dna-and rna-binding proteins by deep learning,” *Nature biotechnology*, vol. 33, no. 8, pp. 831–838, 2015.

[27] X. Bresson and T. Laurent, “Residual gated graph convnets,” *arXiv preprint arXiv:1711.07553*, 2017.

[28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” in *International Conference on Learning Representations (ICLR)*, 2017.

[29] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[30] R. C. Elandt-Johnson and N. L. Johnson, *Survival models and data analysis*. John Wiley & Sons, 1980, vol. 110.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, “Regularization of neural networks using dropconnect,” in *International conference on machine learning*, 2013, pp. 1058–1066.

[33] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[34] Y. Gal and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning,” *arXiv preprint arXiv:1506.02142*, 2015.

[35] C. Vignac, G. Ortiz-Jiménez, and P. Frossard, “On the choice of graph neural network architectures,” in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8489–8493.

[36] F. Errica, M. Podda, D. Bacciu, and A. Micheli, “A fair comparison of graph neural networks for graph classification,” *arXiv preprint arXiv:1912.09893*, 2019.

[37] G. Li, C. Xiong, A. Thabet, and B. Ghanem, “Deepergcn: All you need to train deeper gcns,” *arXiv preprint arXiv:2006.07739*, 2020.

[38] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. Bronstein, and F. Monti, “Sign: Scalable inception graph neural networks,” *arXiv preprint arXiv:2004.11198*, 2020.

[39] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in *International Conference on Machine Learning*. PMLR, 2017, pp. 1263–1272.

[40] K. Kong, G. Li, M. Ding, Z. Wu, C. Zhu, B. Ghanem, G. Taylor, and T. Goldstein, “Flag: Adversarial data augmentation for graph neural networks,” *arXiv preprint arXiv:2010.09891*, 2020.

[41] R. Brossard, O. Frigo, and D. Dehaene, “Graph convolutions that can finally model local structure,” *arXiv preprint arXiv:2011.15069*, 2020.